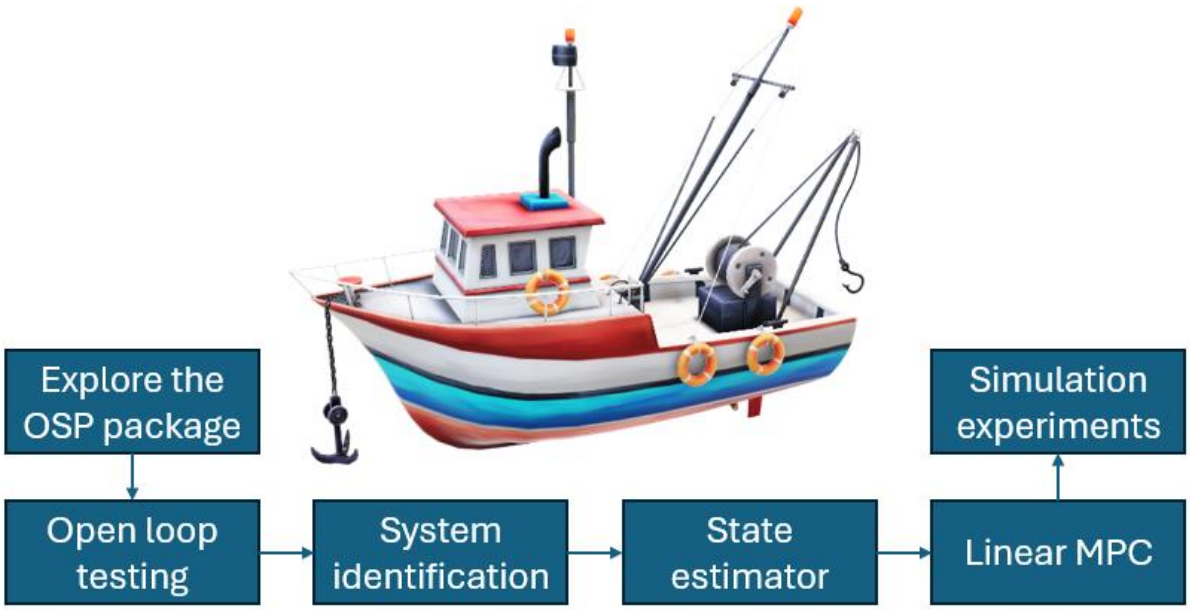
University of South-Eastern Norway

www.usn.no

FMH606 Master's Thesis 2024

Industrial IT and Automation

Dynamic Positioning, system identification and control of marine vessels - using the OSP simulator



Jan-Robin Brustad

Faculty of Technology, Natural sciences and Maritime Sciences

Campus Porsgrunn

University of South-Eastern Norway

www.usn.no

Course: FMH606 Master's Thesis, 2024

Title: Dynamic Positioning, system identification and control of marine vessels - using the OSP simulator

Number of pages: 107

Keywords: Open Simulator Platform, MIMO system, Model predictive control, System identification, State estimation, qpOASES, MATLAB and Simulink.

Student: Jan-Robin Brustad

Supervisor: David Di Ruscio

External partner: None

Summary:

This project explores the Balchen model and the OSP package with a goal to find a vessel model through system identification and try to construct a DP system.

The objective of this project is to perform literature research about DP systems for marine vessels, do system identification of an existing model and implement a DP system for the selected marine vessel.

The methods chapter focuses on open loop testing to try to find the boundaries of the system, system identification in Simulink, and the development of an MPC controller in Simulink. Simulink was used to perform “live” simulations, while MATLAB was used to initialize Kalman filter, and set up NE diagrams showing the marine vessels position.

For the MPC controller an LQ optimal control problem was selected as a start, and this had to be fit in the standard QP formulation that was supported by qpOASES solver.

Finaly, the conclusion discusses how successful this project was in regards of the objective.

Preface

This project stemmed from the curiosity to explore the OSP package, explore the possibilities it brings in regards of system identification and the development of an MPC controller for a marine vessel.

This report follows the IMRaD structure with the introduction divided into introduction and theory. Here the reader will gain beginner friendly theory about DP systems and the different parts needed to design a controller with the OSP package, MATLAB and Simulink, and how to perform different experiments.

The successful goal for this project is to be able to present an MPC controller designed for one of the elements in the OSP package.

Lastly, I would like to thank my supervisor David Di Ruscio for the support and guidance throughout this project. His feedback on my work has been of great importance and has helped me navigate through a whole new field in my knowledge. I would also express my gratitude to my partner and family for supporting me throughout my work and helping me keep my eyes on the end goal, the final delivery.

Porsgrunn, 15.05.2024

Jan-Robin Brustad

Contents

Nomenclature/Abbreviation

Table 1-1 Abbreviation list

Abbreviation	Definition	Explanation
DP	Dynamic Positioning	A system that can position a marine vessel at any place and keep it in that position.
D-SR	Deterministic and Stochastic systems and Realization	A method of subspace system identification. It can be used to completely identify a Kalman filter model from known input and output data, as well as identifying the system order.
FMI	Functional Mock-up Interface	FMI is a standardized interface predominantly utilized in computer simulations. It is widely recognized as the preferred standard for co-simulation.
FMU	Functional Mockup Unit	An FMU is a set of models that conforms to the FMI standard. It is commonly used to share models between different simulation environments.
GPS	Global positioning system	GPS is a navigation system based on satellites and it can provide users around the world with precise location and time information.
MCR	Maximum continuous rating	The highest power output that a marine engine can deliver continuously.

MIMO	Multiple Inputs, Multiple Outputs	A system with more than one input and output.
MPC	Model Predictive Control	MPC is a control algorithm that uses a predictive model to optimize future system responses based on given inputs, continually adjusting to achieve the best performance.
NE	North and East	A two-dimensional diagram showing the marine vessel position in north and east.
NED	North, East, and Down.	A diagram that shows the marine vessel position in north, east and down direction.
PID	Proportional Integral Derivative	PID is a type of feedback controller that adjusts system outputs based on the measured error between a desired setpoint and the actual output. The error is processed through three distinct functions, the proportional, integral, and derivative which together adjust the control action to minimize the error.
OSP	Open Simulator Platform	The OSP is an open-source software package for co-simulation of maritime vessels. The platform builds on the FMI standard and aims to reuse digital twins.
SP	Set point	The desired value for a process that an operator has selected.
SSM	State space model	A mathematical framework that can be used to predict the future states of a system and filter the output values.

1 Introduction

In this chapter a brief introduction to dynamic positioning is given, the objective is elaborated and some of the previous work is discussed. There is also presented a short system overview which elaborates how this work differentiates from the previous work, and the report structure is explained at the end.

1.1 Background

Marine vessels are complex and can be very challenging to maneuver close to oil platforms or other marine installations. A typical marine vessel consists of actuators like tunnel thrusters in the front and the back, a propeller, and a rudder. To dock a marine vessel precise control of these actuators is needed and this is where dynamic positioning comes in handy.

Dynamic positioning systems where first developed in the early 60's in the USA[1] and the development soon escalated around the world for more advanced systems. The dynamic positioning systems in Norway were pioneered by Jens A. Balchen who traveled to USA to study their DP systems[1]. He later co-wrote an article that made the foundation for DP systems in Norway and influenced Kongsberg to take advantage of DP systems in 1975[1].

One of the big problems with classical PID controllers is the need for deviation for the controller to give a control value to the actuators. For marine vessels multiple PID controllers are needed, and each of them needs to be tuned individually. Jens A. Balchen solves this problem by implementing a modern control algorithm based on the Kalman filter[2]. The Kalman filter is especially useful since it can predict unmeasured states of the system that can be utilized in a control algorithm like the MPC controller.

A DP system for ships can includes autopilot and automatic positioning without using an anchor.

The background for this project is the interest to reconstruct the DP systems from the Balchen model through system identification and evaluate the need for a modified version. To reach this goal a model in the OSP package can be selected to perform system identification on, instead of a real ship.

1.2 Objective

The objective is divided into different tasks:

- Perform literature research about DP systems of ships.
- Use an existing dynamic model of a marine vessel and perform experiment design for system identification in order to create models of the vessel.
- Implement a DP system for the vessel.
- Perform Simulation experiments by using MATLAB or similar.

The objective is further elaborated in Appendix A.

1.3 Previous work

The previous work related to this project is spread around multiple topics. The first and maybe most important work is the Balchen et al 1980 model[2]. DP systems where first developed with PID controllers in the early sixties. There were one PID controller for each surge, yaw and sway motion [2]. This obviously has some disadvantages because of coupling between sway, surge, and yaw. This means that the integral action of the controllers must be slow in order for the controllers to give proper control value. Further on in the Balchen model a successful controller consisting of modern control algorithms, the Kalman filter and optimal control is introduced[2].

In 2022 Nour Mohamad Bargouth developed a control system based on the Balchen model. Bargouth tried implementing several different controllers, all of them based on MPC. The types of MPC's explored where standard MPC, reduced size MPC, simple MPC and simple MPC with integral action[3].

1.4 System overview

This project differs from the previous work in the model selection's part, here there is taken an experimental approach to finding a model using the OSP package. The OSP package comes with a demo selection of maritime reference models[4], and the Gunnerus-DP was selected since it came with an ready to go marine vessel with thruster dynamics. The reason for this is to have the opportunity to do system identification on a model that is as realistic as possible.

The developed systems are divided into 4 parts, one part for open loop testing, one for system identification, one for designing a state estimator and a final part where an MPC controller is developed for the selected marine vessel.

The development tools that were selected are MATLAB and Simulink, where MATLAB is used for scripting, state estimatior initializing and for making NE diagrams which is very difficult in Simulink. Simulink is, however, a solid software for executing a simulation in real time and makes it easy for the operator to adjust the different set points and watch live changes.

1.5 Methods

This project is divided into multiple parts that are listed below.

- Open loop testing.
- System identification
- Developing a state estimator.
- Developing an MPC controller.
- Developing an MPC controller with integral action.

The reason for dividing the project into those parts is to create a barrier between the different systems developed. This way, one part is finished when the next part is started and can later be run individually as a standalone system.

The open loop testing focuses on testing the selected marine vessel and seeing how it reacts to inputs. It is also used to explore what a realistic thrust in the main propeller would look like.

The system identification builds on the open loop test application but focuses more on making a setup to create random input signals to the different actuators and measure the outputs.

The development of a state estimator also builds on the open loop setup but focuses on running a state estimator of the marine vessel in parallel with the real model to measure the accuracy of the identified state space model.

The development of a controller mainly focuses on an MPC controller, but PID controller is also discussed. Further on the MPC controller is explored more in depth, and the implementation of integral action is considered.

1.6 Report structure.

The report is structured into 6 main parts, which includes the introduction, theory, methods, results, discussion and finally a conclusion. The purpose of the introduction chapter is to give the reader a picture of the work and methods used.

In the theory chapter the reader gains some backgrounds knowledge about different topics such as PID, MPC, DP, OSP and more. This theory part is useful when reading this report if some of the subjects are unknown from before.

The methods and result chapters are divided into two parts; the methods chapters elaborate the methods and strategies used to reach the objective, and the result chapter explains and presents the results.

The discussion chapter analyses the results regarding problems or success with the given models, methods, and if there are any deviations. Finally, the conclusion summarizes the report and the work done to figure out if it is in line with the objective of this project.

2 Theory

The following chapter presents relevant theory that will be considered when choosing a method.

2.1 Dynamic positioning

For a marine vessel to have a DP system, there must be a control system that controls surge, sway, and yaw. By controlling these variables, it is possible to keep a marine vessel at a stationary location. This is especially useful when trying to dock a supply ship close to an oil rig or other fixed positions[2].

A dynamic positioning system usually consist of a control algorithm, positioning measurement, a propulsion and a rudder system[2]. The control algorithm can either be a PID controller for each surge, sway, and yaw or an MPC controller that controls them all at the same time. The controller algorithm can be considered the heart of the DP system and is crucial for a marine vessel to keep a position and maintain safety measures for the ship.

The positioning measurement can consist of multiple sensors and models. Typically, a GPS system is implemented, but there can also be other systems. The positioning reference system can either be relative positioning, or absolute positioning system[5]. The GPS system is an absolute positioning system, while a relative positioning system can be based on laser technology and gives the position in relation to a target.

The propulsion system usually consists of different thrusters and propellers. The main propeller controls the surge, and sway can be controlled using tunnel thrusters or azimuth thrusters[5]. Finally, yaw can be controlled using a rudder or even an azimuth thruster. Figure 2-1 shows example of placement for the different thrusters, propeller, and rudder on a marine vessel.

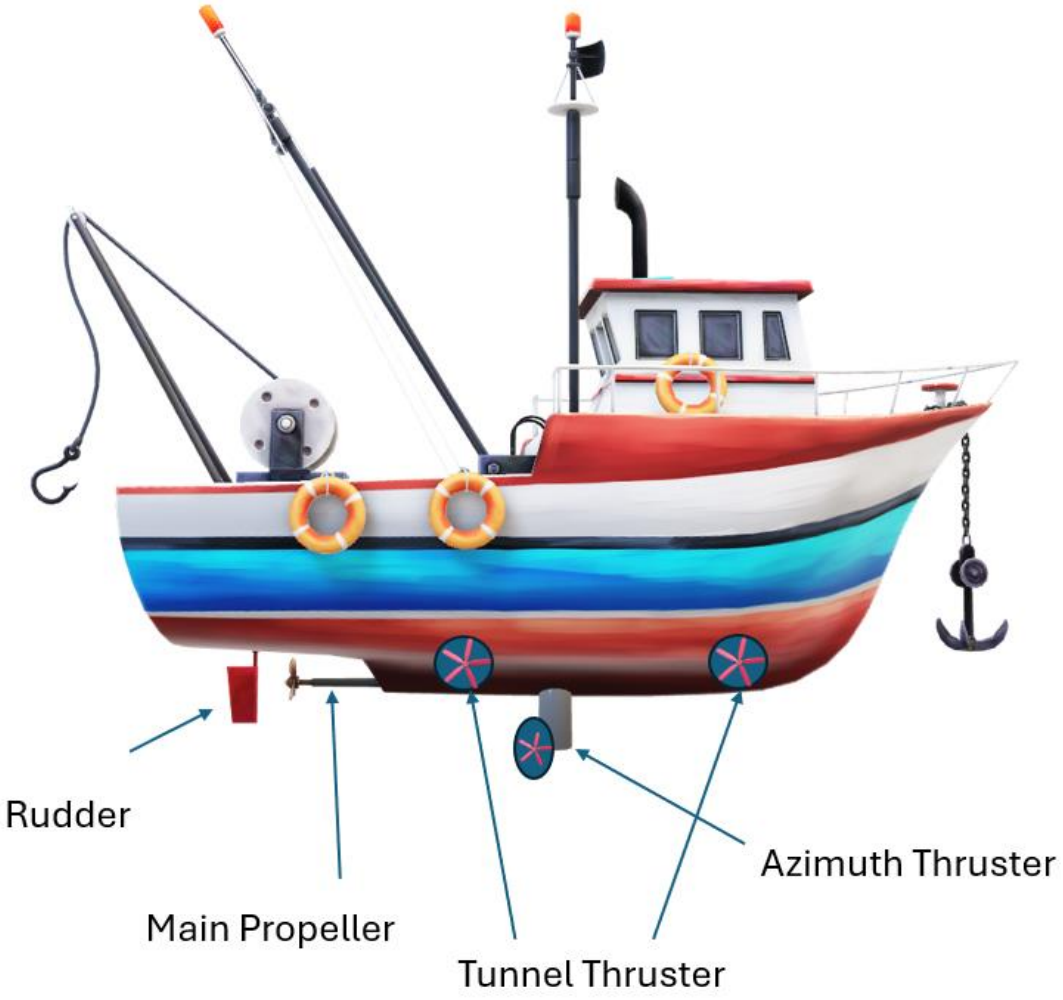


Figure 2-1 Thrusters, propeller and rudder illustrated on a marine vessel.

2.1.1 Six Degrees of freedom

Figure 2-2 illustrates the six degrees of freedom that a marine vessel has. They are called yaw, sway, heave, roll, surge, and pitch [6], surge, sway and heave are linear measurements in meter and yaw, roll and pitch are angles measured in rad. Only three of them are directly controllable and those are surge, yaw, and sway. Typically, a propeller or a thruster controls surge which can be thought of as forward or backwards direction. Sway is sideway control and is typically done through tunnel thrusters. The yaw is a rotational movement that decides how the marine vessel turns. A rudder usually decides how yaw is controlled.

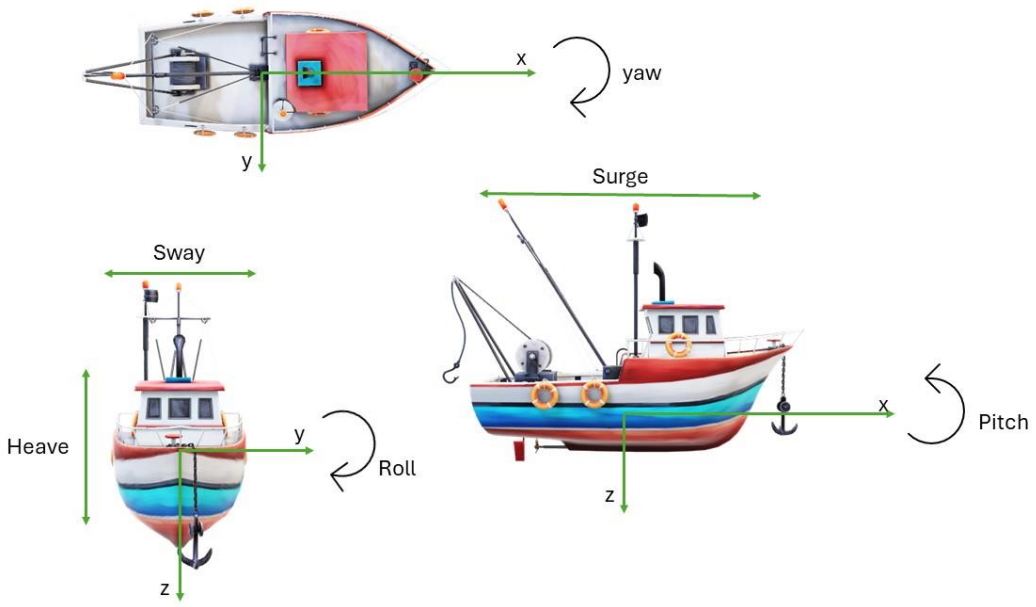


Figure 2-2 A marine vessel with six degrees of freedom.

2.1.2 Input, output and predicted states.

For this system there are three inputs, three outputs and three unknown states that can be predicted. In all there are six states in the model which is illustrated in Figure 2-3. The disturbances, if implemented, could also be thought of as an input and could be included in the system identification part.

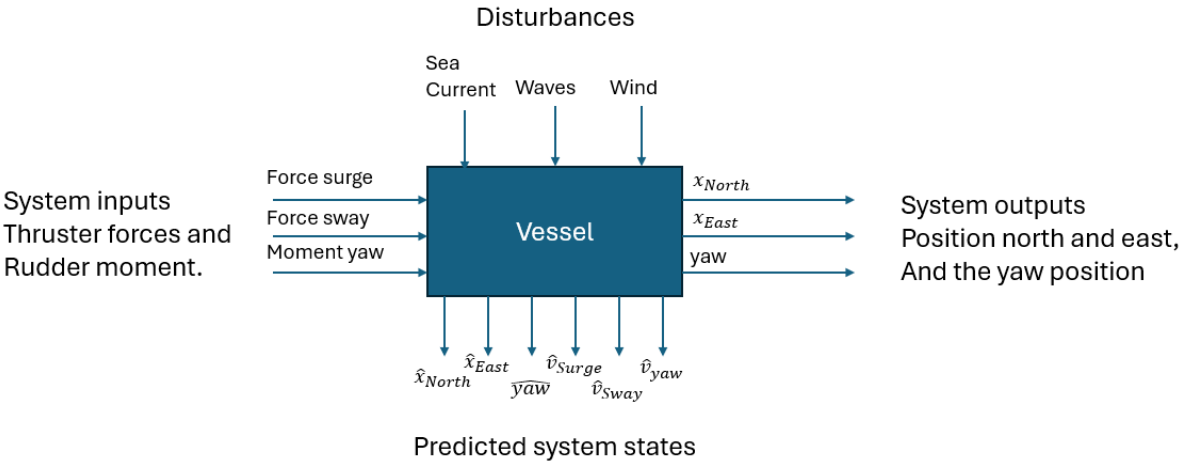


Figure 2-3 Inputs, Outputs, Disturbances, and states of the system.

2.2 Forces acting on a marine vessel.

For this project three different disturbances are considered: The sea current, waves, and wind forces. These disturbances are neglected for the development parts of this project, but they are still relevant and will be discussed briefly in this chapter.

2.2.1 Wind

A marine vessel is subject to wind forces in all degrees of freedom, here only the wind forces in surge, sway and yaw will be analyzed. Figure 2-4 shows how the wind can affect surge, sway, and yaw. If the wind acts with greater pressure on either the rear or front of the marine vessel it will influence the marine vessel to rotate in the yaw direction. This means that the rudder, or azimuth thrusters need to compensate for the acting force.

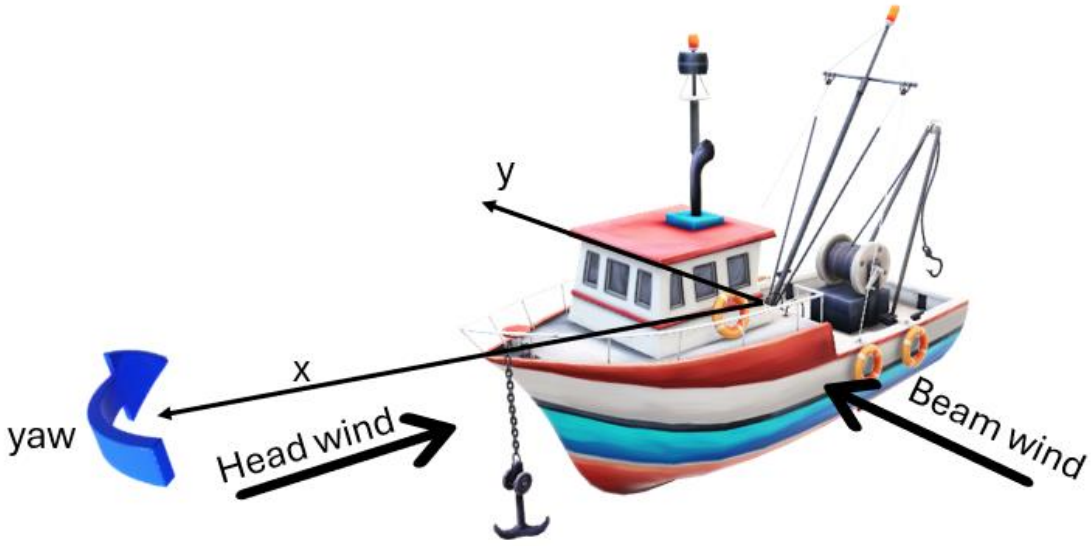


Figure 2-4 Wind force acting on marine vessel, in BODY coordinates. Here head wind is affecting surge direction, and beam wind is affecting sway direction.

The combined wind force can be expressed like equation (2.1) for symmetric marine vessels at rest[7].

$$F_w = \begin{bmatrix} F_{w,su} \\ F_{w,sw} \\ N_w \end{bmatrix} = \frac{1}{2}\rho_a V_w^2 \cdot \begin{bmatrix} C_x \cos(\gamma) A_F \\ C_y \sin(\gamma)\, A_L \\ C_N \sin(2\gamma)\, A_L L \end{bmatrix} \tag{2.1}$$

Here are the variables explained:

F_w: The total force of the wind given as a matrix.

$F_{w,su}$: The wind force in surge direction [N].

$F_{w,sw}$: The wind force in sway direction [N].

N_w: The wind force in yaw [Nm].

ρ_a: The air density of the wind. 1.23[kg/m^3][3].

V_{rw}: Relative wind speed [m/s].

C_x: The wind coefficient in surge direction. $C_x \in [0.50, 0.90]$ [7].

C_y: The wind coefficient in sway direction. $C_y \in [0.70, 0.95]$ [7].

C_N: The wind coefficient in yaw. $C_N \in [0.05, 0.20]$ [7].

γ: The angle of the wind affecting the marine vessel.

A_F: The frontal projected area of the wind on the marine vessel. This is illustrated as head wind in Figure 2-4.

A_L: The lateral projected area of the wind on the marine vessel. This is illustrated as beam wind in Figure 2-4.

L: The overall length of the marine vessel.

The wind angle of attack can be expressed as equation (2.2)[7].

$$\gamma = \psi - \beta v_w - \pi \tag{2.2}$$

Here,

ψ: The angle of the wind relative to the vessel heading in surge.

βv_w: The wind direction relative to the true north [rad].

π: Shifting the wind angle by 180 degrees, changing the wind from where it is coming to where the wind is heading.

Equation (2.3) shows the relative wind speed[7].

$$V_{rw} = \sqrt{v_{rw,su}^2 + v_{rw,sw}^2} \tag{2.3}$$

Where:

$v_{rw,su}$: The component of the relative wind speed in the surge direction.

$v_{rw,sw}$: The component of the relative wind speed in the sway direction.

$$v_{rw,su} = V_w \cos(\beta v_w - \psi) \tag{2.4}$$

$$v_{rw,sw} = V_w \sin(\beta v_w - \psi) \tag{2.5}$$

For equation (2.4) and (2.5):

V_w: is the measured wind speed given in [m/s].

The wind speed and description of the wind is given in Table 2-1[7].

Table 2-1 Wind speed in knots

Description of wind	Wind speed [knots].
Calm	0-1
Light air	2-3
Light breeze	4-7
Gentle breeze	8-11
Moderate breeze	12-16
Fresh breeze	17-21
Strong breeze	22-27
Moderate gale	28-33
Fresh breeze	34-40
Strong gale	41-48
Whole gale	49-56
Storm	57-65
Hurricane	More than 65

To calculate the wind speed in m/s, the formula in equation (2.6) can be used.

$$1[knots] = 0.51[m/s] \quad (2.6)$$

2.2.2 Waves

A marine vessel is influenced by waves in a first order and a second order part. This means that the waves exist of a low frequency part, and a higher frequency oscillatory part. For simulation purposes it is a good idea to separate those two[7].

Wave-frequency motion: The first order wave force component gives a zero mean oscillatory motion. This can be removed by introducing a filter to the wave forces[7].

Wave drift forces: The second order wave force component is a slow varying component which is often referred to as the low frequency wave motion. To counter this low frequency part of wave force integral action must be used[7].

2.2.3 Sea current

The Balchen model introduces the sea current in NED coordinates that includes the current velocity. The sea current is given in equation (2.12), (2.13), and (2.14)

$$\dot{v}_{c,N} = \eta_{c,N} \tag{2.7}$$

$$\dot{v}_{c,E} = \eta_{c,E} \tag{2.8}$$

$$\dot{N}_{c,\psi} = \eta_{c,\psi} \tag{2.9}$$

Here:

$\eta_{c,N}, \eta_{c,E}, \eta_{c,\psi}$: zero mean white noises

$v_{c,N}$: Water current velocity in the north direction.

$v_{c,E}$: Water current velocity in the east direction.

$N_{c,\psi}$: Water current moment in yaw

To transform the sea current to BODY coordinates the transpose transformation matrix can be used from equation(2.10) [2].

$$R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.10}$$

Water current velocities in BODY coordinates are then given in equation (2.11).

$$\begin{bmatrix} v_{c,surge} \\ v_{c,sway} \\ N_c \end{bmatrix} = R(\psi)^T \begin{bmatrix} v_{c,N} \\ v_{c,E} \\ N_{c,\psi} \end{bmatrix} \tag{2.11}$$

Here:

$v_{c,surge}$: Water current velocity in surge direction.

$v_{c,sway}$: Water current velocity in sway direction.

N_c : Water current moment in yaw.

2.3 Vessel models

A vessel model can either be a mathematical model [2], a real marine vessel or a simulated version of a marine vessel. An example of an simulated vessel can be the Gunnerus-DP [8] provided as a reference model in the OSP package. In the following subchapters the Balchen model and Gunnerus models are given which will lay the foundation for the project work.

2.3.1 Balchen model

The Balchen model is a mathematical model describing a marine vessel motion [2]. The model is designed to be used with a Kalman filter making it suitable for MPC controllers. To understand the basics behind the Balchen model, one would have to understand Newtons second law.

$$\sum F = ma \tag{2.12}$$

Where:

F – Forces acting on the vessel [N].

m – The mass of the vessel [kg].

a - The acceleration of the vessel [m/s^2].

Equation (2.12) can be further expanded where the sum of forces is wind force, current force from water and thruster force.

$$\sum F = F_w + F_c + F_T \tag{2.13}$$

Here:

F_w – The wind force [N].

F_c – The current force from sea water [N].

F_T – The thruster force [N].

Since the high frequency part of the Balchen model is very small and oscillates around the low frequency part[2] it is neglected for this project. The low frequency model is given by equation (2.14) to (2.19). For easier readability the same variable letters and indexes have been used as in the previous work by Bargouth [3].

$$\frac{dx_{su}}{dt} = v_{su} \tag{2.14}$$

$$\frac{dx_{sw}}{dt} = v_{sw} \tag{2.15}$$

$$\frac{d\psi}{dt} = v_{\psi} \tag{2.16}$$

$$v_{su} = -\frac{d_1}{m_1}\left|v_{su} - v_{c_{su}}\right|\left(v_{su} - v_{c_{su}}\right) + \frac{1}{m_1}\left(F_{w_{su}} + F_{t_{su}}\right) + \eta_1 \tag{2.17}$$

$$v_{sw} = -\frac{d_2}{m_2}\left|v_{sw} - v_{c_{sw}}\right|\left(v_{sw} - v_{c_{sw}}\right) + \frac{1}{m_2}\left(F_{w_{sw}} + F_{t_{sw}}\right) + \eta_2 \quad (2.18)$$

$$v_{\psi} = -\frac{d_3}{m_3}\left|v_{\psi}\right|v_{\psi} - \frac{d_4}{m_3}\left|v_{sw} - v_{c_{sw}}\right|\left(v_{sw} - v_{c_{sw}}\right) + \frac{1}{m_3}\left(N_c + N_w + N_t\right) + \eta_3 \quad (2.19)$$

The different variables are defined below:

x_{su}: The position of the marine vessel in x direction (surge) [m].

v_{su}: The velocity of the marine vessel in x direction [m/s].

x_{sw}: The position of the marine vessel in y direction (sway) [m].

v_{sw}: The velocity of the marine vessel in y direction(sway) [m/s].

ψ: The heading of the marine vessel in yaw [rad].

v_{ψ}: The yaw velocity in heading of the marine vessel [rad/s].

η_1, η_2, η_3: These are assumed to be from zero mean gaussian white noise process[2].

d_1, d_2, d_3, d_4: These are the drag and momentum coefficients[2]. The difference in angle between the vessel heading and water current direction is usually given as a function, where the drag and momentum coefficients can be gathered.

m_1, m_1, m_1: These are the inertial coefficients which can be assumed to be constants[2].

$F_{w_{su}}$: This is the wind force in x direction.

$F_{w_{sw}}$: This is the wind force in y direction.

$F_{t_{su}}$: This is the thruster force in x direction.

$F_{t_{sw}}$: This is the thruster force in y direction.

N_c, N_w, N_t: This are the current moment, wind moment and thrust moment.

$v_{c_{su}}, v_{c_{sw}}$: Current in x direction and current in y direction.

2.3.2 R/V Gunnerus

The R/V Gunnerus is a marine research vessel owned by NTNU and was first deployed in 2006 for its purposes. R/V Gunnerus is named after Johan Ernst Gunnerus and is the second ship named Gunnerus by NTNU[9].

The Gunnerus-DP package comes with several FMUs that can be used. This includes the vessel model, and the vessel model is shown in Figure 2-5.

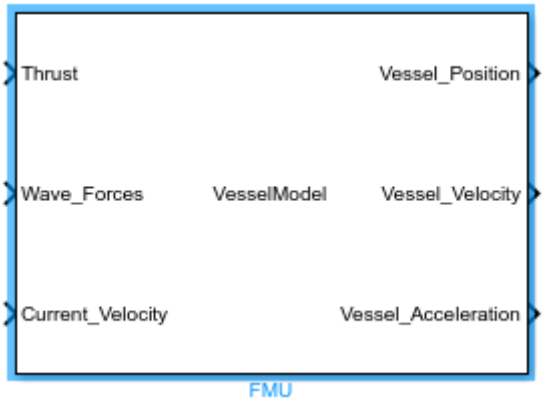


Figure 2-5 Vessel model imported as an FMU in Simulink.

The sum of forces has been split into thrust, wave forces and current velocity. This is similar to the Balchen model as explained in chapter 2.3.1. The FMUs in the Gunnerus-DP package is depicted in Figure 2-6.

FMU name	Description
Box Reference	Provides setpoints for a box maneuver
Control System Communication	Network FMU handling communication between control system and simulator
Current Model	Current model generating surface current velocities
DP Controller	3-Dof DP controller
Reference Model	Reference model providing setpoints to DP
Simulator Communication	Network FMU handling communication between control system and simulator. Shown in Figure 2 as FmiUdpAdapter.
Thruster Dynamics	Thruster dynamics
Vessel Model	6-dof vessel model

Figure 2-6 Gunnerus-DP FMU packages description[8]

As Figure 2-5 shows there are no wind forces connected to the vessel model. The documentation for the Gunnerus-DP states that additional forces can be connected to either the Wave_force, or the Thrust in Figure 2-5 since they are both internally connected to the same summation block [8].

2.4 OSP

OSP stands for Open Simulation Platform and is an open-source software for simulating marine vessels[10]. It has the possibility to co-simulate marine equipment and even complete marine vessels like the Gunnerus-DP.

The OSP package comes with many software applications, including a demo application with a user interface and a command prompt-based co-simulator(libcosim). There are also many reference models that can be freely used[4].

The OSP packages are built on the FMI standard meaning it can be integrated into many different software's, including MATLAB with Simulink. The Gunnerus-DP has been included in the reference model package and the vessel model can be imported as an FMU into Simulink[8].

2.5 MATLAB and Simulink

For this project MATLAB version 9.12.0.2327980 (R2022a) Update 7 and Simulink version 10.5 (R2022a) is used. Both MATLAB and Simulink are products of MathWorks and are accessible from MathWorks.com[11]. MATLAB serves primarily as a scripting environment, and Simulink offers a graphical programming environment designed for modeling and simulating. They work great together and often an initializing script can be made in MATLAB for a simulation in Simulink. Furthermore, Simulink offers the ability to log data directly to MATLAB workspace that can be fetched and stored in matrices or vectors. This makes it possible to make graphs and analyze data in the MATLAB environment after a Simulink simulation.

They both support extra packages outside the standard installation and are backed by a big community that provides a lot of learning resources. An example of this is the YouTube channel for MATLAB[12].

2.6 FMI and FMU

The OSP simulator depends on FMUs of different models to function. FMU can be the model of a marine vessel, thruster dynamics or even simple physics laws for simulation purposes. The main reason for having the FMI standard in the OSP simulator is to co-simulate. This means that different frameworks can use the same FMU model and interact with it at the same time. The reason they can communicate with the same FMU model lies in the FMI standard. FMI stands for Functional mock-up interface and is an interface that tells how software like MATLAB can communicate with FMUs. In Figure 2-7 it is shown how MATLAB with Simulink and a OSP simulation is both communicating to the same FMU model.

Figure 2-7 How MATLAB with Simulink and the OSP simulation can interact with the same FMU over the FMI standard.

2.7 Kalman Filter

The Kalman Filter is an algorithm for state estimation and prediction, and it is quite good at estimating states that are not easily measurable. It is a model-based algorithm which requires identifying a model of the system through system identification or mathematical modelling. It can estimate states that are not measured, and the states that are measured. Typically the Kalman filter runs in parallel with the real process to predict states that are not measured for various reasons[13]. The reasons can be that it is expensive to install sensor equipment, or that the state can't be measured directly, for example the temperature in a rocket chamber of a spacecraft.

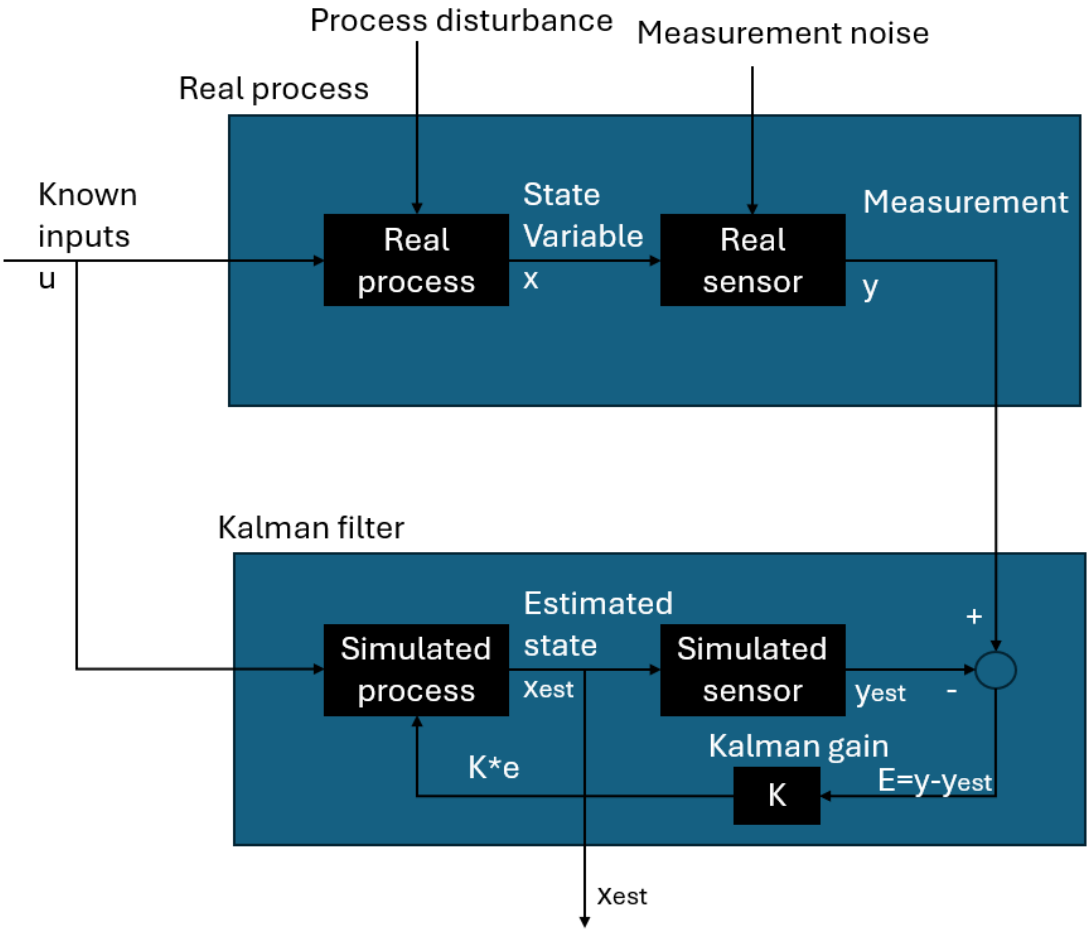


Figure 2-8 Kalman Filter and real process illustrated together.

In Figure 2-8 y_{est} is shown, this is the filtered value of y, the process value and sometimes it is better to use this if the measurements are prone to noise. The user interface can for example minimize ripple effects making graphs easier to read for the operator.

The formulas used in the Kalman filter is shown in equation (2.20) and (2.21), this are the SSM formulas.

$$x_{k+1} = Ax_k + Bu_k + f_k(\bar{v}_k) \tag{2.20}$$

$$y_k = Cx_k + g_k(\bar{w}_k) \tag{2.21}$$

Here:

x_{k+1}: The next predicted state.

A: State transition matrix.

x_k: The current state value.

B: The input matrix.

u_k: The input vector at current timestep.

y_k: The output value at current timestep.

C: The output matrix.

$f_k(\bar{v}_k)$: Process noise or disturbance

$g_k(\bar{w}_k)$: the measurement noise or disturbance in output.

2.8 Controller

There are many types of controllers that can be used to control a marine vessel. In this report two types of controllers are considered.

2.8.1 MPC

MPC stands for Model Predictive Control and is a controller type that uses a model of a system to predict future behavior. The reason for this is that the real system can't be calculated in advance, but the model can[14].

MPC is an algorithm that solves an optimization problem. The optimization problem is the control value and needs to be solved at each timestep. The reason for solving the control optimization problem at each timestep is that new measurement data is available[14].

An MPC controller solves the control value for many steps forward but uses only the first control value. The rest is discarded since the control optimization problem must be solved again[15]. MPC can use a sliding horizon strategy where the initial state is used to calculate the first control value, then the system takes one timestep forward and uses the previous calculated state values to calculate a new control value[14].

An MPC controller can handle MIMO systems very efficiently. This means that they can handle multiple control values for different actuators for example.

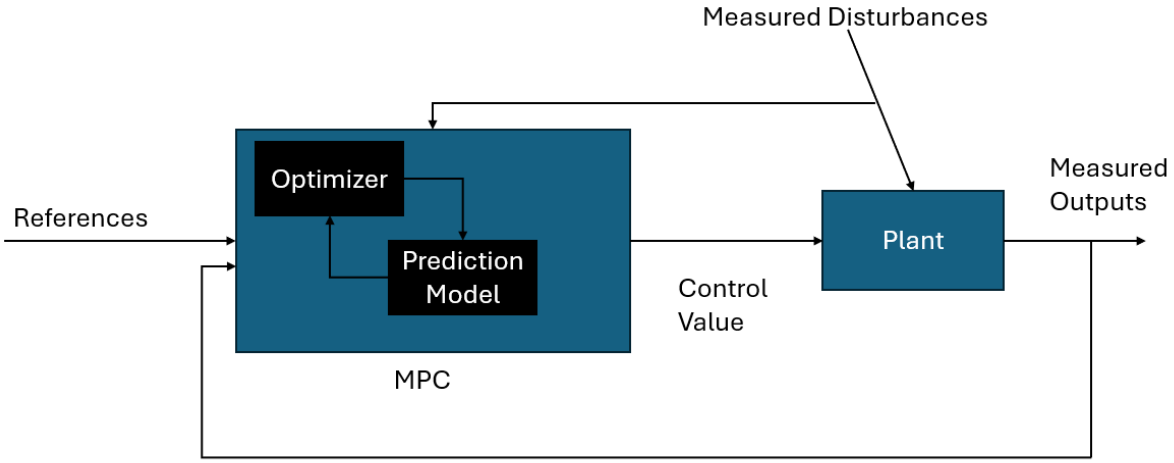


Figure 2-9 Block diagram for a basic MPC controller.

Figure 2-9 shows the block diagram for a basic MPC controller. The controller consists of a prediction model, for example the Kalman filter and an optimizer.

To use an MPC controller in Simulink a solver must be imported. qpOASES is an solver that is supported by Simulink and is relative easy to set up if the developer has some general knowledge in adding an C++ compiler in MATLAB, there are some instructions on this from Roshan Sharma website in the course "Model Predictive Control, IIA4717"[16].

2.8.2 PID

A PID controller is a controller that is dependent on the error to give a control value. PID stands for Proportional, Integral and Derivative. PID controllers are the main controllers of choice because of their simplicity. A PID controller can be split into simpler parts, for example P controller, PI controller, PID controller or a PD controller[17].

Since the PID controller is dependent on the error, the error must be calculated continuously. The error is the difference between the reference value and the measured value as seen in formula (2.22).

The PID controller is excellent for SISO models but can also be used for MIMO systems. For MIMO systems there must be a PID controller for each the surge, sway, and yaw.

$$e(t) = r(t) - y(t) \tag{2.22}$$

Here:

$e(t)$: The error signal, representing the error between reference value and measurement.

$r(t)$: The reference value, often referred to as SP which is the state the control system aims to achieve.

$y(t)$: The actual measurement value, typically from a sensor.

$$u(t) = K_p \cdot e(t) + \frac{K_p}{T_i} \cdot \int_0^t e(t) \cdot dt + K_p T_d \cdot \frac{d\, e(t)}{dt} \quad (2.23)$$

$u(t)$: The control signal that is applied to the system, based on the error.

K_p: The proportional gain, this a tuning parameter that is multiplied to the error.

T_i: The integral time, used to calculate the integral gain $\frac{K_p}{T_i}$.

T_d: The derivative time. Used to calculate the derivative gain $K_p T_d$.

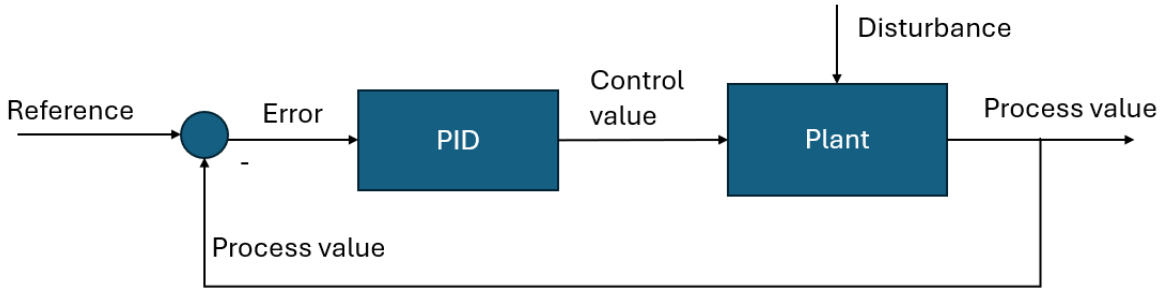


Figure 2-10 Schematic view of the PID controller.

2.9 System identification

To make a mathematical model of the marine vessel, system identification is essential. As illustrated in Figure 2-11 the process of system identification is shown. Different values are inserted to the system's inputs, and then the outputs are measured. This data is used to make a state space model through realization theory[18].

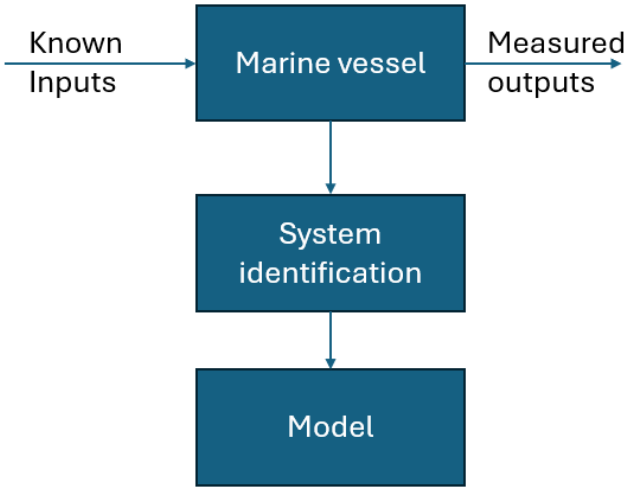


Figure 2-11 System identification principle. Here both the inputs and outputs must be logged.

For a marine vessel the inputs consist of thrusters, wind, sea current, and wave forces. The outputs can consist of vessel position, the vessels velocity, and the vessels acceleration. The chosen model is assumed to be a state space model.

The known inputs and measured outputs can be logged in to matrices, U for inputs and Y for outputs.

The inputs consist of thrusters and may include waves, wind, and sea current. In this report only the thruster inputs are considered and put in the input matrix U as shown in equation (2.24).

$$U = \begin{bmatrix} U_{Surge,1} & U_{sway,1} & U_{\psi,1} \\ U_{surge,2} & U_{sway,2} & U_{\psi,2} \\ \vdots & \vdots & \vdots \\ U_{surge,N} & U_{sway,N} & U_{\psi,N} \end{bmatrix} \tag{2.24}$$

The measured outputs are the position in north, east, and yaw. There is also possible to measure the velocity and acceleration in surge, sway, and yaw but this is not done in this report. The output matrix Y is given in equation (2.25).

$$Y = \begin{bmatrix} Y_{north,1} & Y_{east,1} & Y_{\psi,1} \\ Y_{north,2} & Y_{east,2} & Y_{\psi,2} \\ \vdots & \vdots & \vdots \\ Y_{north,N} & Y_{east,N} & Y_{\psi,N} \end{bmatrix} \tag{2.25}$$

In equation (2.24) and (2.25) N stands for the number of samples.

3 Methods

The following chapter focuses on the methods used for solving the objective in this project. The first objective literature research is done through the theory chapter, and this method chapter focuses mainly on practical work.

The marine vessel selected to conduct experiments on is the Gunnerus ship from the OSP package[8]. This package comes with both vessel model and a thruster dynamics model as FMUs. The reason for choosing this model is the lack of a real model, and it gives opportunities to explore the OSP package.

3.1 Open loop testing.

To test how the vessel model in Gunnerus-DP responds to step changes in surge, sway and yaw a Simulink program was made. The open loop testing program in Simulink is shown in Figure 3-1.

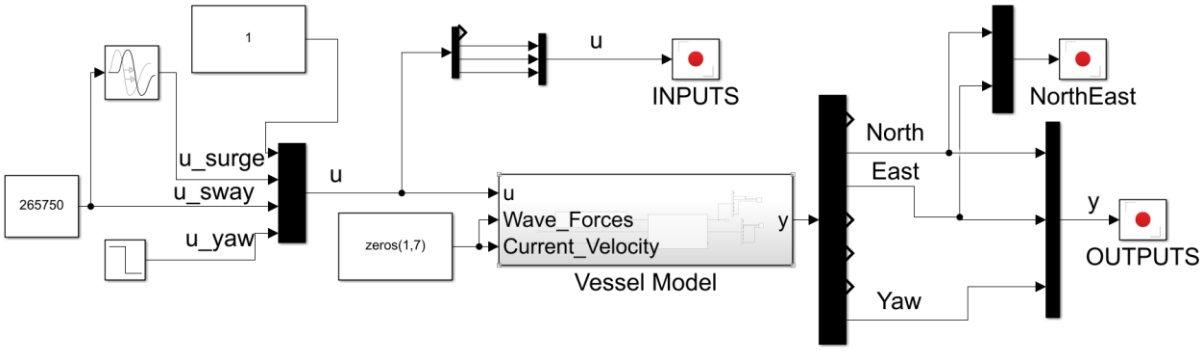


Figure 3-1 Open loop testing in Simulink with only thruster force acting on the marine vessel.

As Figure 3-1 shows there are used different approaches to generate an input signal to the vessel model. Here experiments with just a constant value, a step response and time delayed signals are tested. The results are evaluated in a NE diagram in chapter 5.1. This is a two-dimensional representation of the marine vessel's position as seen from above.

The constant block with the value 1 shown in Figure 3-1 does nothing. ThrusterDynamics should have only three inputs, but an extra index 0 is added which doesn't seem do anything, hence the use of an extra constant block. More on this in the discussion chapter. The ThrusterDynamics as well as the VesselModel FMUs are shown in Figure 3-2.

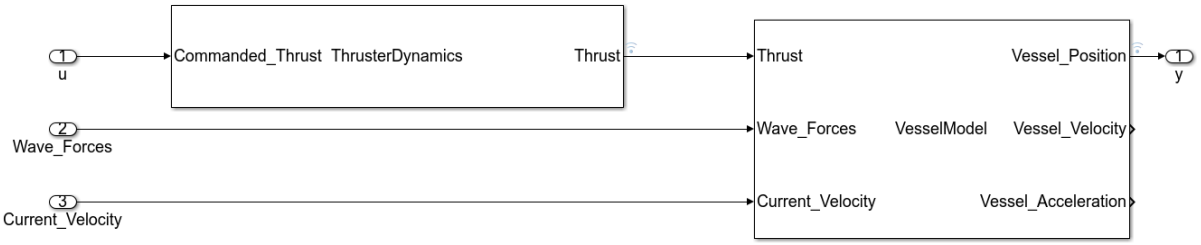


Figure 3-2 Thruster dynamics and vessel model imported as FMUs; this is the Vessel Model subfunction from Figure 3-1.

The ship's response to both positive and negative control inputs in the surge and yaw, as well as in surge and sway directions, was tested to determine the necessary forces for realistic vessel movement.

During the experiments a force of 100000N was selected to give a realistic output force in regards to the datasheet[19] of R/V Gunnerus. Here the propeller's horsepower is considered as well as the weight of the ship. Due to the absence of detailed specification of the azimuth thrusters and the rudders, their forces have been assumed to be equivalent to the main propeller. This approximation was necessary to proceed with the experiment, despite the lack of precise information on the forces these components generate.

The thrust that the main propeller gives can be calculated by the equation in (3.1).

Here:

- T is thrust.
- P is power in watts.
- v is speed in m/s.
- η is the propulsion efficiency.

$$T = \frac{P}{v \cdot \eta} \tag{3.1}$$

With an educated guess of the propulsion efficacy of $\eta = 0.55$ from figure 2.02 in "Basic principles of ship propulsion[20], the calculation of thrust is as follows.

$$T = \frac{1000000W}{6.842 \cdot 0.55} = 265750N \tag{3.2}$$

3.2 System Identification

This chapter describes the system identification process of the marine vessel Gunnerus. To identify a SSM there must be generated some input data and some measurements of the output data. Since there is provided an FMU model in the OSP package for the Gunnerus marine vessel, the input data and the measurements are recorded from the usage of this model. This FMU model is depicted in Figure 3-2.

To construct the SSM from the gathered input and output data the D-SR algorithm[21] is used. The D-SR algorithm presents the matrices for the SSM, and it is considered a ready to use discrete time linearized model.

3.2.1 Input data and measurements.

The input data is generated through a random source block and is then sent through a MATLAB function. The reason for this is that the random source block gives a single number between -1 and 1, for example 0.7. This is not a clean square wave input, and the MATLAB function(fcn) simply converts values over 0.0 to 1 and values equal to and below 0.0 to -1. This part of the program is shown in Figure 3-3. Notice that the signal is multiplied by a constant, and this is just to make a more realistic thrust that can move the ship as discussed in 3.1. The selected value of force must be able to move the ship that is about 72t[19].

The reason for making square wave inputs is that it is not sufficient to only estimate a model with a simple step response. A robust model demands an experiment with thruster forces in all available thrusters to capture the best possible model. It might also be needed to run the same experiment multiple times to find a more accurate model that represents the system behavior.

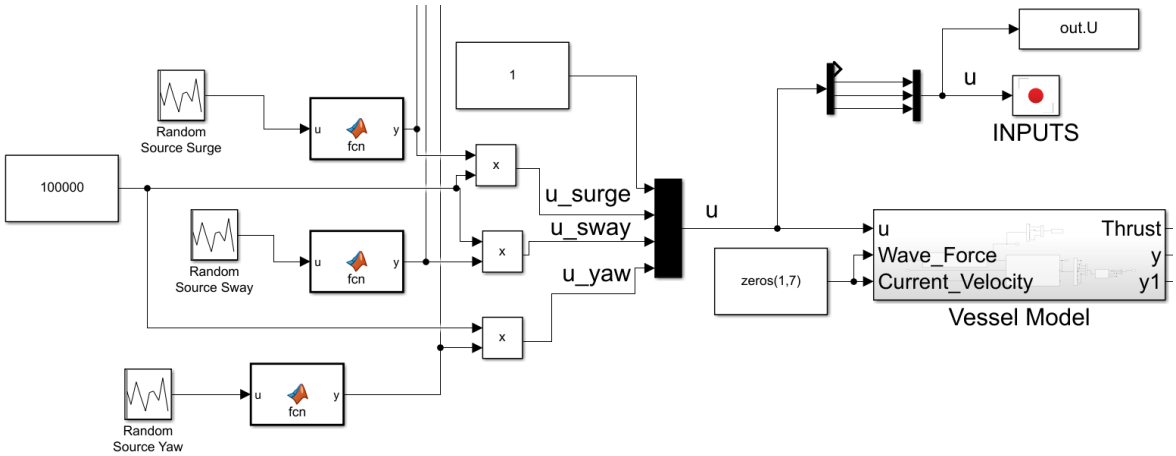


Figure 3-3 A Simulink program for generation of input data in surge, sway, and Yaw.

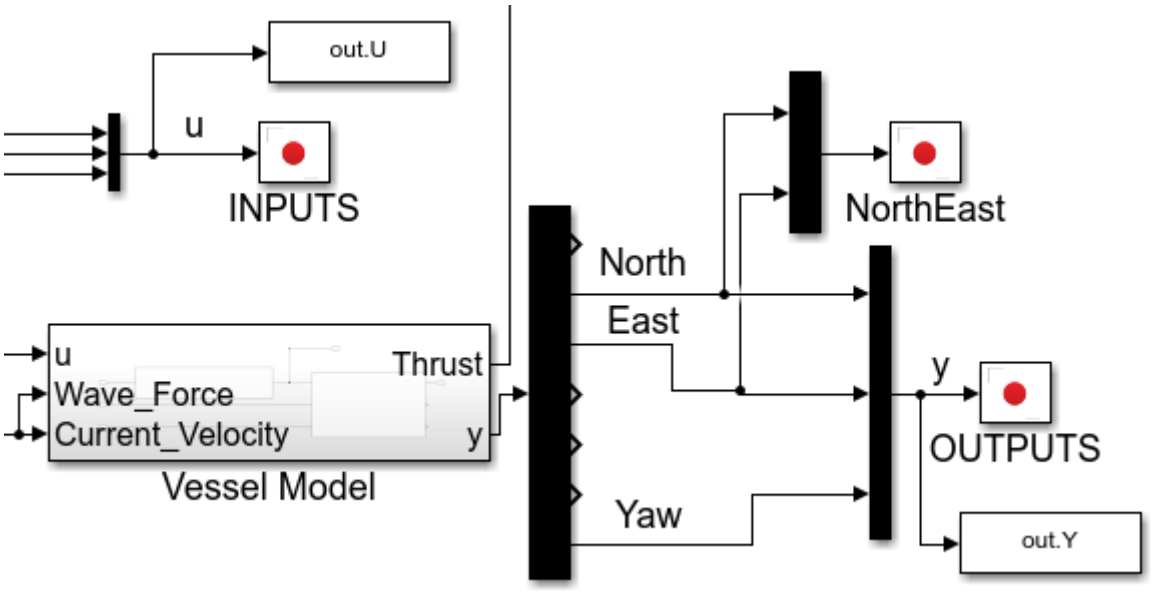


Figure 3-4 Measuring the outputs from the Vessel Model in position North, East, and Yaw.

When both the out.U and out.Y blocks have recorded the signals, they are now available in MATLAB workspace. Figure 3-5 shows how the data is made available from the logging through a MATLAB script.

```
1   %Inputs
2   SurgeInput = out.U(:,1);
3   SwayInput = out.U(:,2);
4   YawInput = out.U(:,3);
5
6   %Outputs
7   positionNorth = out.Y(:,1);
8   positionEast = out.Y(:,2);
9   positionYaw= out.Y(:,3);
10
```

Figure 3-5 accessing input and output data from MATLAB workspace.

The input matrix U and output matrix Y can now be created as shown in Figure 3-6. The complete code is shown in appendix H.

```
11
12      %Check that all vectors are of same length
13      if length(SurgeInput)==length(SwayInput) && length(SurgeInput)==length(YawInput) ...
14              && length(SurgeInput)==length(positionNorth) && ...
15              length(SurgeInput)==length(positionEast) && ...
16              length(SurgeInput)==length(positionYaw)
17
18
19          %Making input matrix
20          U = [SurgeInput, SwayInput, YawInput];
21          %Making output matrix
22          Y = [positionNorth, positionEast, positionYaw];
23
24      else
25
26          error("Data vectors are of different lenght");
27
```

Figure 3-6 Creating input and output matrix. This was done several times until a sufficient SSM system was identified.

3.2.2 Using D-SR to get an SSM.

When the input matrix U and output matrix Y is found after the experiment it is time to use the D-SR function in MATLAB[21]. It is a ready to use function that will make the SSM based on matrix Y and U. Equation (3.3) shows the MATLAB command that is used to find the matrices A, B, C, D, CF, F and x0.

$$[A, B, C, D, CF, F, x0] = dsr(Y, U, L) \tag{3.3}$$

Here:

A is the state transition matrix.

B is the input matrix.

C is the output matrix

D is the direct transmission matrix.

x0 is the initial values.

L is the identification horizon used to predict the number of states. If the user knows the number of states, the suggestion from the D-SR algorithm can be skipped and just enter det number of states.

CF and F is related to the Kalman filter gain and can be calculated as equation (3.4) shows[21].

$$K = CF \cdot F^{-1} \tag{3.4}$$

3.2.3 Analysis of the SSM

When system identification has found a model, it is a good idea to check the stability of the system. The stability of the system can be done by analyzing the eigenvalues of the state transition matrix A. To find the eigenvalues the MATLAB function eig.m can be used.

Since the D-SR algorithm returns a discrete system there are some criteria for the system to be stable.

Stable system: If all eigenvalues are inside the unity circle on the complex plane the system is considered stable. This means that the system can reach a steady state when there is an input to the system. For example, for a marine vessel the input could be increased propeller force.

Unstable system: For an unstable system one or more of the eigenvalues are outside the unity circle. This means that the system is not able to reach a steady state if one or more of the eigenvalues is above 1.

To check the controllability features of the system, the rank of the controllability matrix must be inspected. The controllability matrix is given by equation (3.5), and it is a tool to check if the pair (A, B) is controllable. The system is only controllable if the rank of the controllability matrix is equal to the rank of the system[22]. The rank of the system in this project is equal to the number of states.

$$C_n = [B \quad AB \quad A^2B \quad \cdots \quad A^nB] \tag{3.5}$$

Here, C_n is the controllability matrix.

To check if the system is observable, the rank of the observability matrix must be examined. The system is observable if the rank of the observability matrix is equal to the rank of the system. The observability matrix is given in equation

(3.6), and the rank of the system is equal to the system states[22].

$$O = \begin{bmatrix} D \\ DA \\ DA^2 \\ \vdots \\ DA^{mi \times n} \end{bmatrix} \tag{3.6}$$

3.3 State estimation and Kalman filter.

The linear Kalman filter is based on the linear SSM that was found in the system identification process. The system matrices are given in chapter 5.2.

The Kalman filter can be used to filter the output values, or even predict unknown states. Here the goal is to predict the states v_{surge}, v_{sway}, and v_{psi} as shown in Figure 2-3.

The state estimator made in Simulink is shown in Figure 3-7. The only difference is that the vessel model takes the input in thrust as a vector and the state estimator takes each input

separately. There are also scopes that show the estimated states, the real measured output, and the filtered output value against each other.

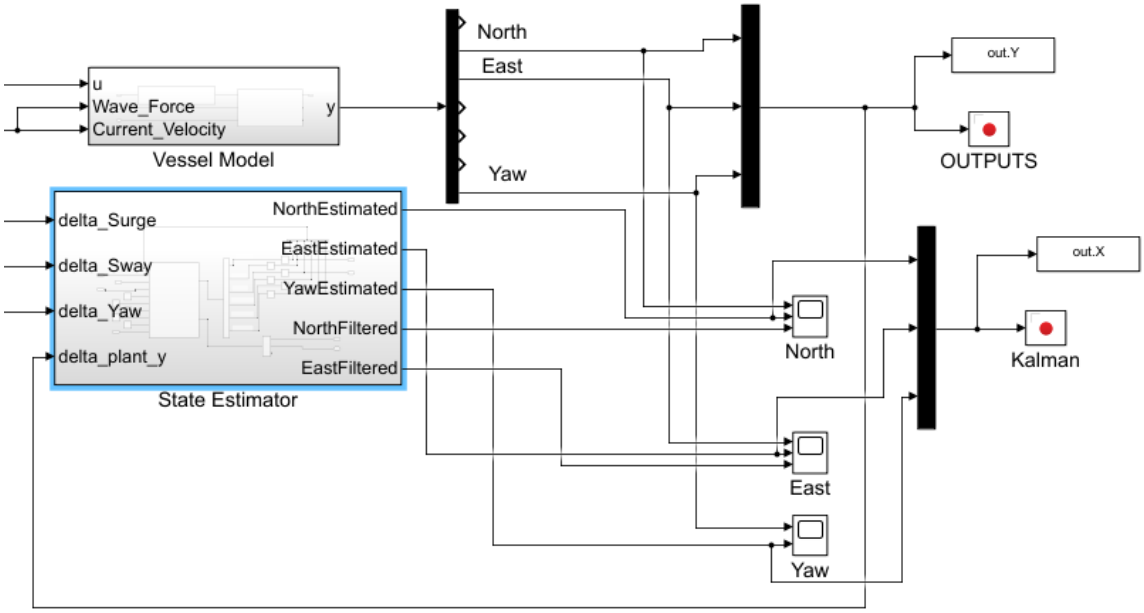


Figure 3-7 State estimator running in parallel with the vessel model. There are also placed scopes in the bottom right to better analyze the values.

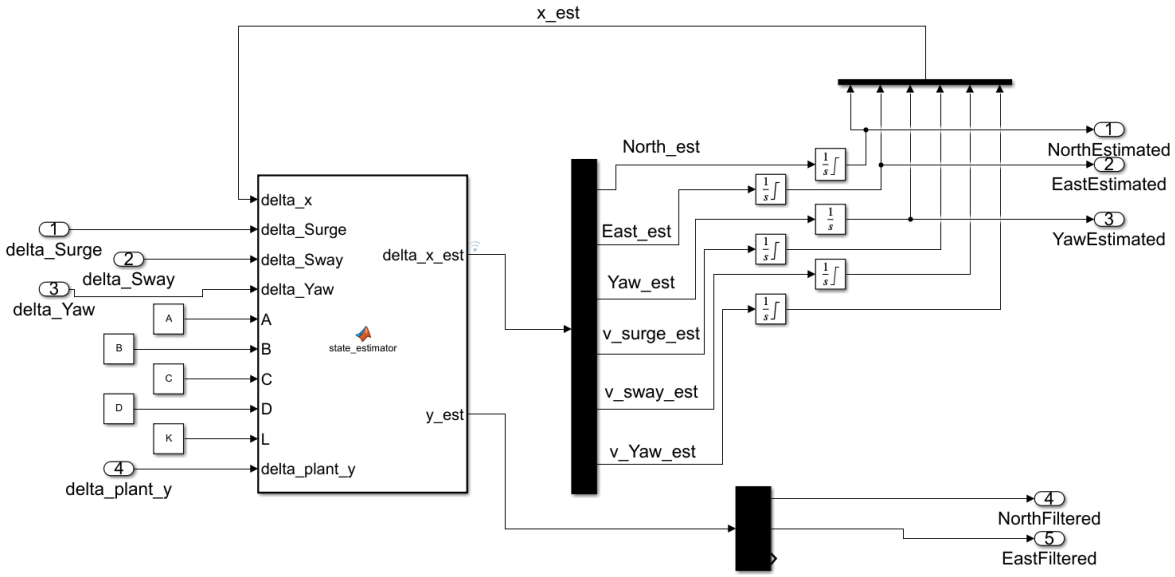


Figure 3-8 The inside of the state estimator based on linear Kalman Filter. This is the sub system “State Estimator” from Figure 3-7. A, B, C, D, and K are imported as constant parameters from the MATLAB workspace.

StateEstimatorComplete ▸ KalmanFilter ▸ MATLAB Function

```
1  function [delta_x_est,y_est]= state_estimator(delta_x, delta_Surge, delta_Sway, delta_Yaw,A, B,C, D,L,delta_plant_y)
2
3  y_est = C*delta_x+D*[delta_Surge;delta_Sway;delta_Yaw];
4  delta_x_est = A*delta_x +B*[delta_Surge;delta_Sway;delta_Yaw] + L*(delta_plant_y-y_est);
5
```

Figure 3-9 MATLAB function for State_estimator in Figure 3-8.

There are also options to differentiate between random input values and manual input values. This is shown in Figure 3-10. This makes it easier to manually enter a step response to check the performance of the state estimator.

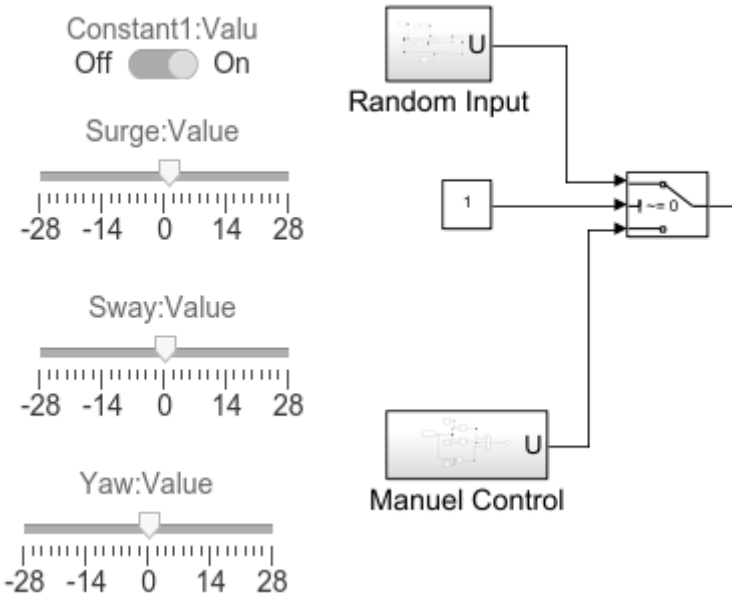


Figure 3-10 Switching option between manual control and random input values. Here an on/off switch is shown in the upper left corner that activates the switching box.

Since the simulation in Simulink happens as fast as the computer manages, it is a good idea to slow down the simulation. This is done using the realtime_pacer_lib which can adjust the simulation so that it's like watching a real ship in action.

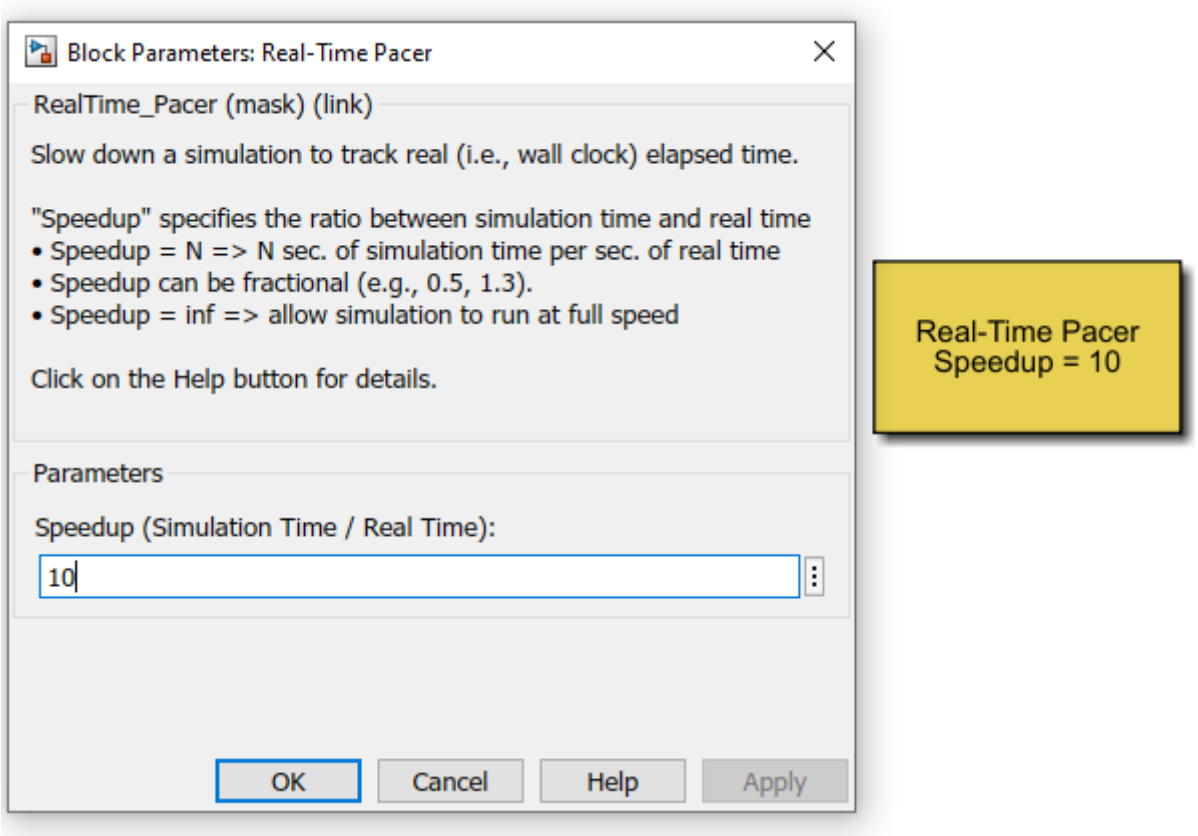


Figure 3-11 Real-Time Pacer block is to be inserted into the Simulink file to slow the simulation down to real time. On the left of the Real-Time Pacer is the configuration page where the relationship between simulation time and real time is given.

3.4 Control of the marine vessels

To make a complete DP system a controller is needed. This following chapter will discuss some types of controllers and result in one type of controller being pursued.

3.4.1 PID Controller

A PID controller might be a good choice in regards of controlling the Gunnerus marine vessel. But a challenge is that the controller would have to consist of multiple different PID controllers. The reason for this is that the PID controller is not suitable for MIMO systems, and the control problem must be described as multiple SISO systems. A PID controller would have to be divided into position control in surge, position control in sway and position control in yaw.

Another problem with the PID controller for an advanced system like this is the lack of information on how the system should behave with different control values for thrusters, disturbances etc. A solution for this could be to use the state estimator developed in chapter 3.3.

3.4.2 MPC Controller

To make an MPC controller for a DP system, the problem can be thought of as an LQ optimal control problem. The LQ optimal control problem can be expressed like equation (3.7).

$$J = \frac{1}{2}[e_1^T Q_1 e_1 + u_0^T P_0 u_0 + e_2^T Q_2 e_2 + u_1^T P_1 u_1 + \cdots + e_N^T Q_N e_N + u_{N-1}^T P_{N-1} u_{N-1}] \tag{3.7}$$

In equation (3.7) N is the prediction horizon, Q is the error weighting matrix for each timestep, and P is the input weighting matrix for each timestep. For both Q and P, it is assumed that they are constant along the whole prediction horizon, meaning $Q_1 = Q_2 = Q$.

The LQ optimal control problem is also subject to:

$$x_{k+1} = Ax_k + Bu_k \tag{3.8}$$

$$y_k = Cx_k \tag{3.9}$$

$$e_k = y_k - r_k \tag{3.10}$$

Since qpOASES doesn't support the standard LQ optimal control problem formulation this has to be converted into the standard QP formulation as equation (3.11) shows. This is the same method for formulating a LQ optimal control problem to QP problem as given in "Lecture Notes for the course IIA 4117: Model Predictive Control"[14].

$$J = \frac{1}{2}\underbrace{\begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix}^T}_{Z^T} \underbrace{\begin{bmatrix} H_{11} & 0 & 0 & 0 \\ 0 & H_{22} & 0 & 0 \\ 0 & 0 & H_{33} & 0 \\ 0 & 0 & 0 & H_{44} \end{bmatrix}}_{H} \underbrace{\begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix}}_{Z} + \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix}^T}_{c^T} \underbrace{\begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix}}_{Z} \tag{3.11}$$

Linear equality constraints: $A_e x = b_e$ (3.12)

Linear inequality constraints: $A_i x = b_i$ (3.13)

Bounds: $x_L \leq x \leq x_U$ (3.14)

The Hessian matrix H is given by equation (3.15).

$$H = \begin{bmatrix} H_{11} & 0 & 0 & 0 \\ 0 & H_{22} & 0 & 0 \\ 0 & 0 & H_{33} & 0 \\ 0 & 0 & 0 & H_{44} \end{bmatrix} \tag{3.15}$$

The different blocks in the Hessian matrix are given below in (3.16), (3.17), (3.18), and (3.19)

$$H_{11} = \begin{bmatrix} P_0 & 0 & \cdots & 0 \\ 0 & P_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P_{N-1} \end{bmatrix}_{N \times N} = I_N \otimes P \tag{3.16}$$

$$H_{22} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}_{N \cdot nx \times N \cdot nx} = I_N \otimes 0_{nx \times nx} \tag{3.17}$$

$$H_{33} = \begin{bmatrix} Q_1 & 0 & \cdots & 0 \\ 0 & Q_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & Q_N \end{bmatrix}_{N \times N} = I_N \otimes Q \tag{3.18}$$

$$H_{44} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}_{N \cdot ny \times N \cdot ny} = I_N \otimes 0_{ny \times ny} \tag{3.19}$$

For the prediction horizon N, the number of unknowns is given by equation (3.20).

$$n_z = N \cdot (n_u + n_x + n_e + n_y) \tag{3.20}$$

Since $n_y = n_e$ equation (3.20) can be formulated as:

$$n_z = N \cdot (n_u + n_x + n_y + n_y) \tag{3.21}$$

Here:

n_u: The number of inputs.

n_x: The number of states.

n_y: The number of outputs.

n_e: The number of errors.

$$c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_N \end{bmatrix} = 0_{n_z \times 1} \tag{3.22}$$

For LQ optimal control problem in this project the inequality equality constraints do not exist. The linear equality constraints are given in equation (3.23) and expanded in equation (3.24) to (3.35).

$$\begin{bmatrix} A_{\varepsilon,1u} & A_{\varepsilon,1x} & A_{\varepsilon,1e} & A_{\varepsilon,1y} \\ A_{\varepsilon,2u} & A_{\varepsilon,2x} & A_{\varepsilon,2e} & A_{\varepsilon,2y} \\ A_{\varepsilon,2u} & A_{\varepsilon,3x} & A_{\varepsilon,3e} & A_{\varepsilon,3y} \end{bmatrix} \begin{bmatrix} u \\ x \\ e \\ y \end{bmatrix} = \begin{bmatrix} b_{e,1} \\ b_{e,2} \\ b_{e,3} \end{bmatrix} \tag{3.23}$$

$$A_{\varepsilon,1u} = \begin{bmatrix} -B & 0 & \cdots & 0 \\ 0 & -B & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & -B \end{bmatrix} = -I_N \otimes B \tag{3.24}$$

$$A_{\varepsilon,1x} = \begin{bmatrix} I & 0 & \cdots & 0 & 0 \\ -A & I & \cdots & 0 & 0 \\ 0 & -A & \ddots & 0 & 0 \\ 0 & 0 & \cdots & -A & I \end{bmatrix} = -I_{N \times n_x} - (I_{N-1} \otimes A) \tag{3.25}$$

$$A_{\varepsilon,1e} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} = 0_{N \cdot n_x \times N \cdot n_y} \tag{3.26}$$

$$A_{\varepsilon,1y} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} = 0_{N \cdot n_x \times N \cdot n_y} \tag{3.27}$$

$$A_{\varepsilon,2u} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} = 0_{N \cdot n_y \times N \cdot n_u} \tag{3.28}$$

$$A_{\varepsilon,2x} = \begin{bmatrix} -C & 0 & \cdots & 0 \\ 0 & -C & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & -C \end{bmatrix} = I_N \otimes C \tag{3.29}$$

$$A_{\varepsilon,2e} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} = 0_{N \cdot n_y \times N \cdot n_y} \tag{3.30}$$

$$A_{\varepsilon,2y} = \begin{bmatrix} I & 0 & \cdots & 0 \\ 0 & I & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & I \end{bmatrix} = I_{N \cdot n_y} \tag{3.31}$$

$$A_{\varepsilon,3u} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} = 0_{N \cdot n_y \times N \cdot n_u} \tag{3.32}$$

$$A_{\varepsilon,3x} = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & 0 \end{bmatrix} = 0_{N \cdot n_y \times N \cdot n_x} \tag{3.33}$$

$$A_{\varepsilon,3e} = \begin{bmatrix} I & 0 & \cdots & 0 \\ 0 & I & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & I \end{bmatrix} = I_{N \cdot n_y} \tag{3.34}$$

$$A_{\varepsilon,3y} = \begin{bmatrix} I & 0 & \cdots & 0 \\ 0 & I & \cdots & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & \cdots & I \end{bmatrix} = I_{N \cdot n_y} \tag{3.35}$$

$$B_{\varepsilon,1} = \begin{bmatrix} Ax_0 \\ 0_{n_x \times 1} \\ \vdots \\ 0_{n_x \times 1} \end{bmatrix} \quad B_{\varepsilon,2} = \begin{bmatrix} 0_{n_y \times 1} \\ 0_{n_y \times 1} \\ \vdots \\ 0_{n_y \times 1} \end{bmatrix} \quad B_{\varepsilon,3} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_N \end{bmatrix} \tag{3.36}$$

$$z_L = \begin{bmatrix} -\infty \\ -\infty \\ \vdots \\ -\infty \end{bmatrix}_{nz \times 1} \quad z_u = \begin{bmatrix} \infty \\ \infty \\ \vdots \\ \infty \end{bmatrix}_{nz \times 1} \tag{3.37}$$

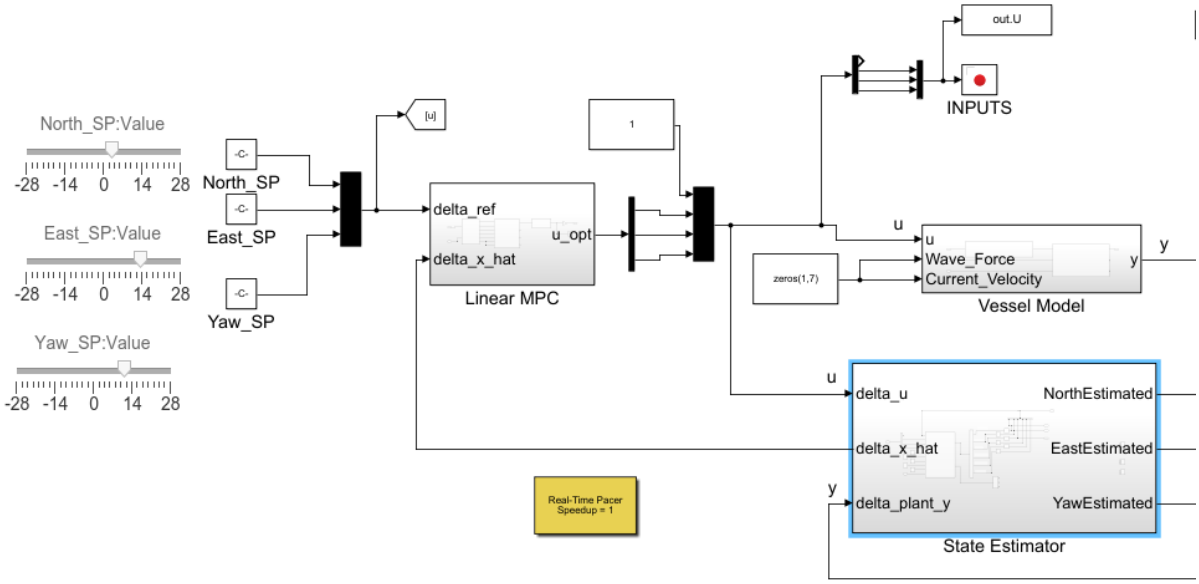


Figure 3-12 The complete linear MPC controller with state estimator. The Real-Time pacer is also implemented which means this run-in real time and it is possible for humans to interact with setpoints.

The complete MPC controller program made in Simulink is presented in Figure 3-12 and Figure 3-13. This program builds on the previous model made in Figure 3-7 for the state estimator. Figure 3-14 shows the subsystem Linear MPC which includes the QP formulation script, the S function and extraction of the first control input.

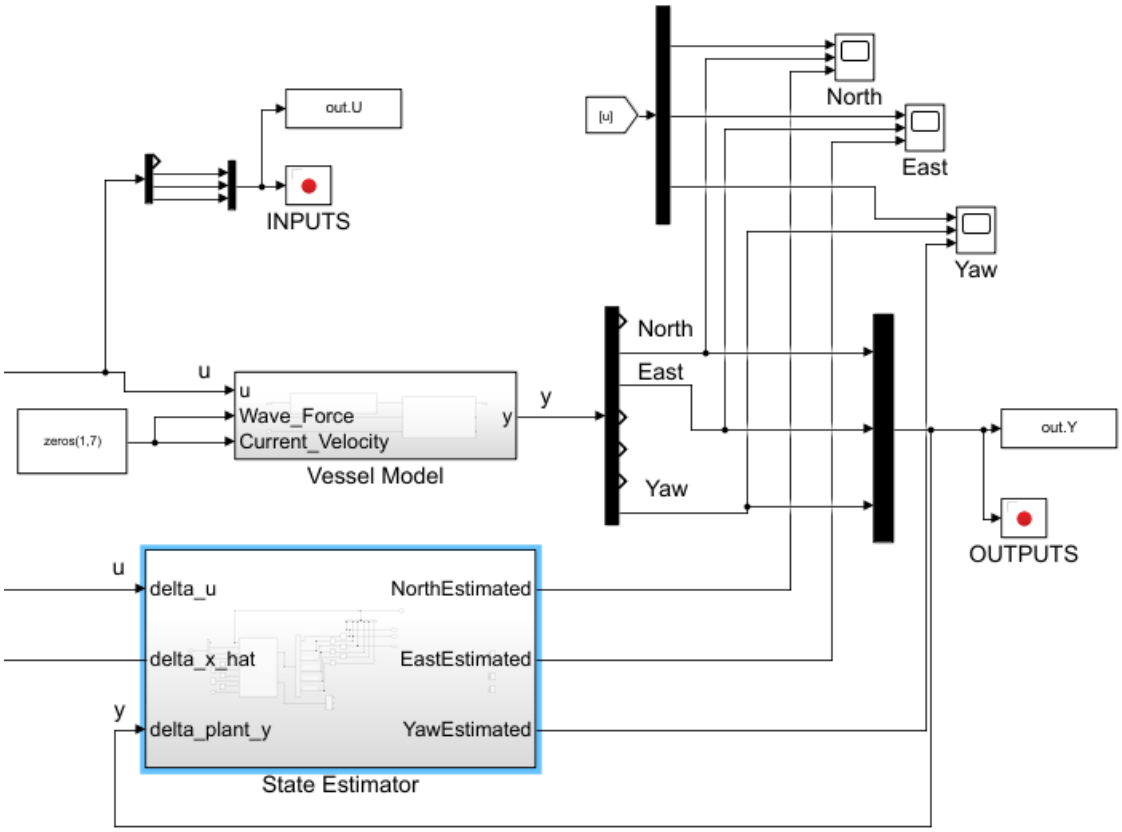


Figure 3-13 This is figure shows the live plotting and logging of data for the MPC controller. This is the right side of the system in Figure 3-13.

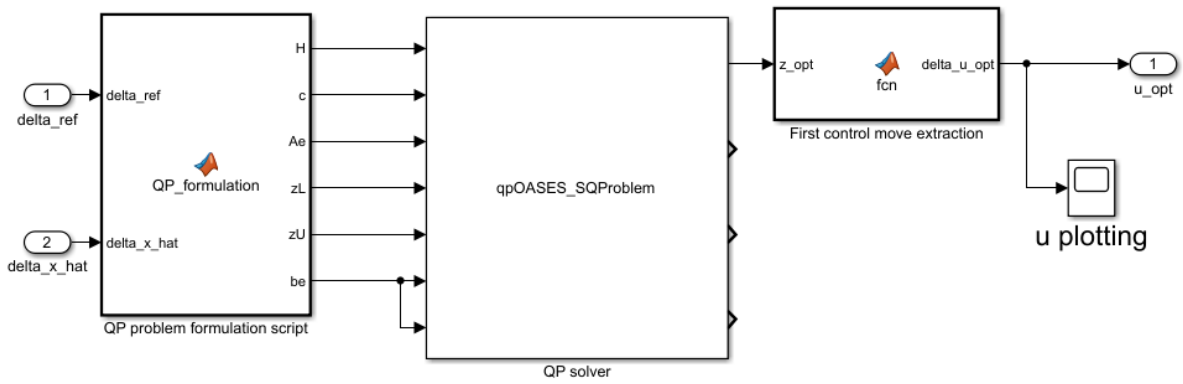


Figure 3-14 This is the inside of the Linear MPC block from Figure 3-12. Here is a MATLAB function for QP problem formulation, and a S-function to implement the qpOASES solver. Only the first control input is needed and is extracted, the rest is discarded.

To use qpOASES the number of unknowns must be known for the whole prediction horizon. If N=20, u=3, x=6, and y=3 the total number of unknowns can be found with equation (3.21) as shown in equation (3.38).

$$n_z = 20 \cdot (3 + 6 + 3 + 3) = 300 \tag{3.38}$$

This is then inserted into the qpOASES source file and compiled to a mex file as shown in Figure 3-15.

```
56  /* SETTINGS */
57  #define SAMPLINGTIME   0.1 /*  -1       */      /**< Sampling time. */
58  #define NCONTROLINPUTS  300 /*2 */              /**< Number of unknowns to optimize. */
59  #define MAXITER         5000                       /**< Maximum number of iterations. */
60  #define HESSIANTYPE     HST_SEMIDEF /*HST_UNKNOWN*/            /**< Hessian type, see docur
61  #define CPUTIME    0 /*max CPU time*/
```

Command Window

```
>> make qpOASES_SQProblem
```

Figure 3-15 A snapshot from the source file of qpOASES, showing the settings part. Here the total number of unknowns must be selected for the specific problem. Afterwards the "make" command must be run in the command window to compile the file.

The compiled file qpOASES_SQProblem file shall then be selected in the S-function called QP solver from Figure 3-14.

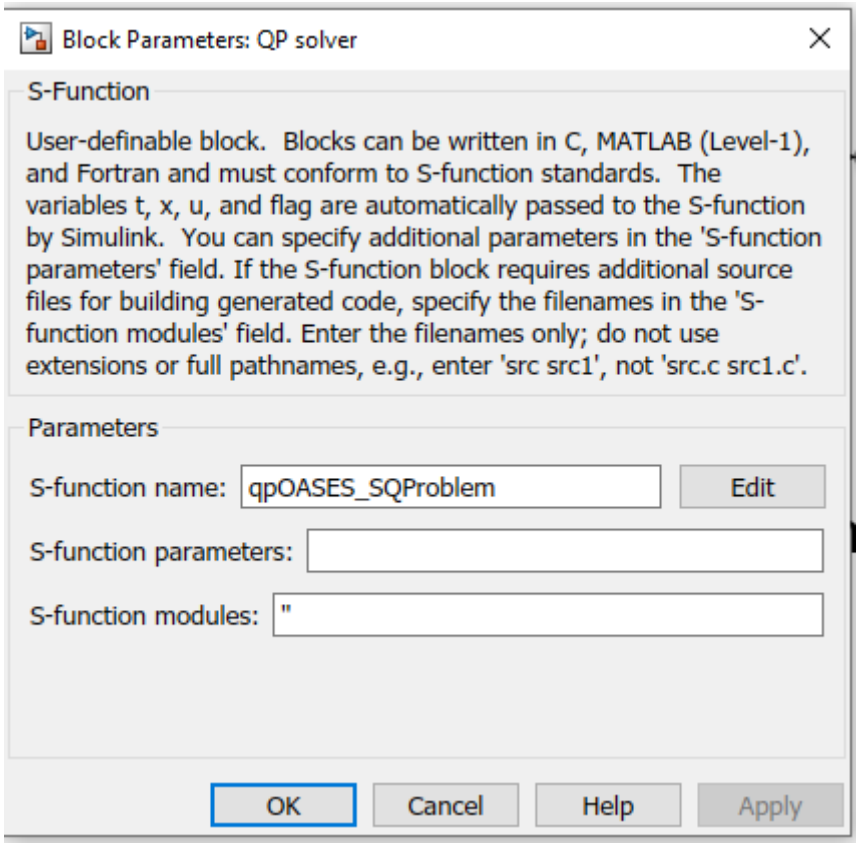


Figure 3-16 S-function that runs qpOASES solver.

Figure 3-17 shows the start of the QP formulation script. The linear equality constraints are implemented in the same way as well as the bounds for the system, more on this in Appendix M.

```
%Discrete time model is imported from
%The system identification part.

N = 20;                          %PRediction horizon length
Q = diag([1e4 1e4 1e4 ]);        %Weighting matrix for the error
%P = 1e-3;                       % Weighting matrix for the input
P = diag([1e-6 1e-6 1e-6]);


x0=delta_x_hat;


r=[ones(1, N).*delta_ref(1);
    ones(1,N).*delta_ref(2);
    ones(1,N).*delta_ref(3)];  %make reference for the whole prediction lenght

%Build the QP problem
%Standard QP formulation
nx = size(A, 1); nu = size(B, 2); ny=size(C, 1); nz = N*(nx+nu+2*ny);

% Building H matrices
H11 = kron(eye(N),P);
H22= zeros(N*nx, N*nx);
H33 = kron(eye(N),Q);
H44 = zeros(N*ny, N*ny);
H_mat = blkdiag(H11,H22,H33,H44);
```

Figure 3-17 A code snippet from the QP formulation script. Here it is shown where the weighting matrix for the error and weighting matrix for the input is set, and the MATLAB implementation of H and nz.

3.4.3 MPC Controller with integral action

To design an MPC controller with integral action, one would have to solve the issue with measurements being in north and east rather than being in BODY coordinates like surge and sway. The reason for this is that it makes more sense to give thrust in surge if the deviation is in surge. To counter this problem a rotational matrix[6] is introduced and given in equation (3.39). Notice that the rotational matrix doesn't include heave as it is not measured in the original system.

$$R(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) \\ \sin(\psi) & \cos(\psi) \end{bmatrix} \tag{3.39}$$

The system is almost equal to Figure 3-12 and Figure 3-13 but there are some deviations regarding the integral action. This is shown in Figure 3-18.

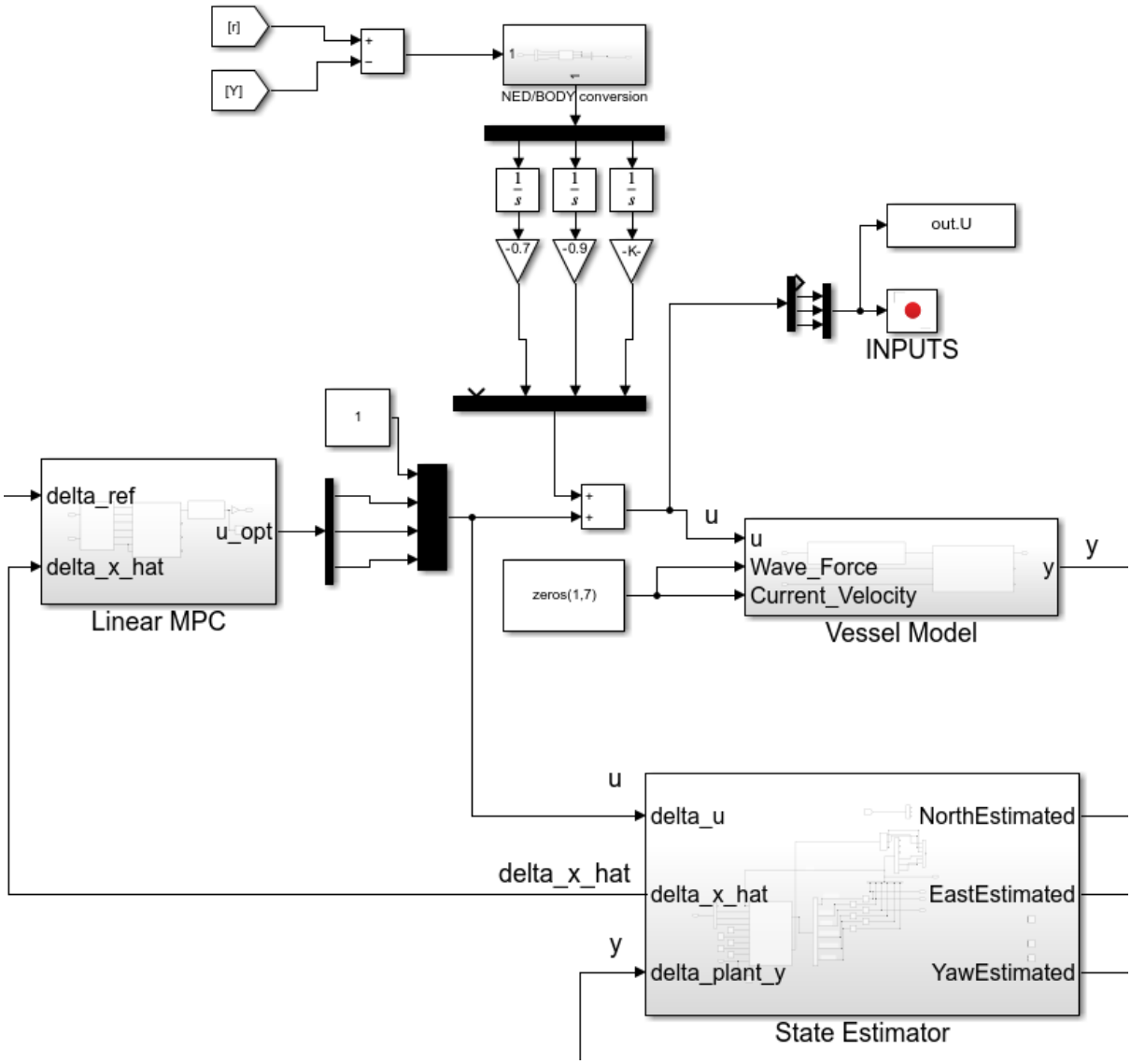


Figure 3-18 Here the integral action is implemented. First the error between the reference r and the measured output Y is calculated, then it is inserted into a NED/BODY conversion block. The signal is then split into surge sway and yaw and integral action is applied.

The gain is found by trial and error and there is a different gain in each surge, sway, and yaw. The sub function NED/BODY conversion is shown in Figure 3-19.

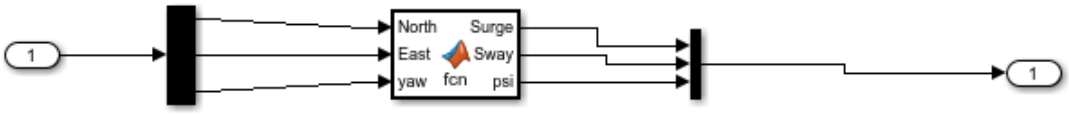


Figure 3-19 Sub function NED/BODY conversion.

The MATLAB function inside Figure 3-19 uses the rotational matrix equation (3.39) to find the coordinates in surge, sway, and yaw.

```
1   function [Surge, Sway, psi]= fcn(North, East, yaw)
2
3   %North - coordinates in north
4   %East - coordinates in east
5   %yaw - angle in radians
6
7   %Correcting variable name
8   psi = yaw;
9
10  %rotational matrix without Down
11  R_psi = [cos(psi), -sin(psi);
12      sin(psi), cos(psi)];
13
14  %NED coordinates
15  P_NED = [North;East];
16
17  %Computing Body coordinates
18  P_body = R_psi*P_NED;
19
20  %Extract Surge and Sway components.
21  Surge = P_body(1);
22  Sway = P_body(2);
23
```

Figure 3-20 Practical use of the rotational matrix from NED to BODY coordinates.

4 Requirements and Design

To make a DP system for a ship some requirements for the system must be collected. This chapter presents some software engineering methods to collect the requirements for the system and generate a design for the final system.

4.1 FURPS+

The requirements for this system can be collected using FURPS+. The complete analysis is given below.

Functional requirements:

- It should be possible to change the set point in surge, sway, and yaw.
- The controller should be able to compute control action based on the current state of the system and the predicted states.

Usability requirements:

- The control system should have an intuitive and easy to understand user interface.

Reliability requirements:

- The DP system shall be able to handle control in surge, sway, and yaw at the same time.

Performance requirements:

- The system should be quick to update to a new set point.
- The system should be able to handle disturbances like wind, sea current and waves.

Supportability requirements:

- The DP system should be easy to upgrade for future needs.

+

- There should be a version control system of the software like GIT.
- A readme file is to be released with the finished product if released on GITHUB.

4.2 UML diagrams

Figure 4-1 shows a simple use case diagram that illustrates how the control system should work and how an operator would interact with it. The control system should read sensor input and adjust outputs accordingly to difference in SP and output. There should also be possible for user interaction like changing the different SP.

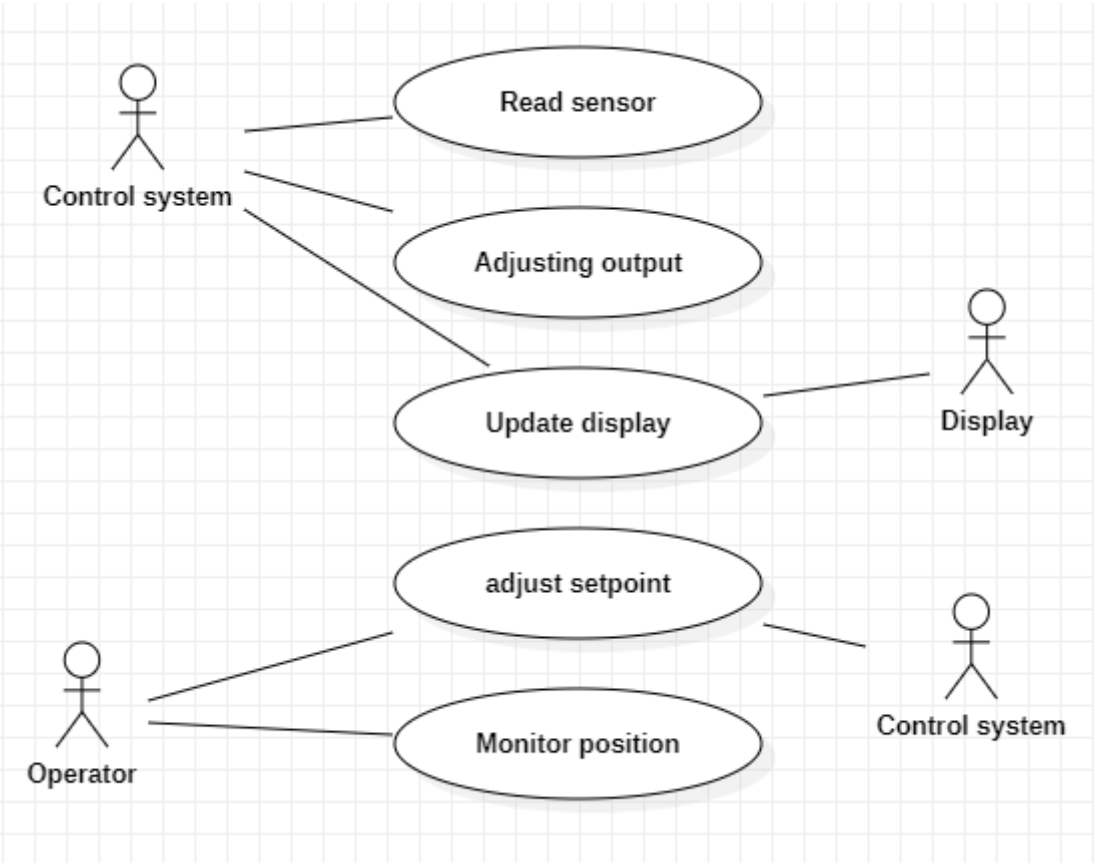


Figure 4-1 A simple use case diagram to better visualize the use cases and actors for the program.

Figure 4-2 shows a simplified system sequence diagram. The goal for this diagram is to give a visual representation of how the program should work. Here an operator can start the control system, and even adjust the SP. The control system will after that maintain the selected SP the best it can through a control algorithm.

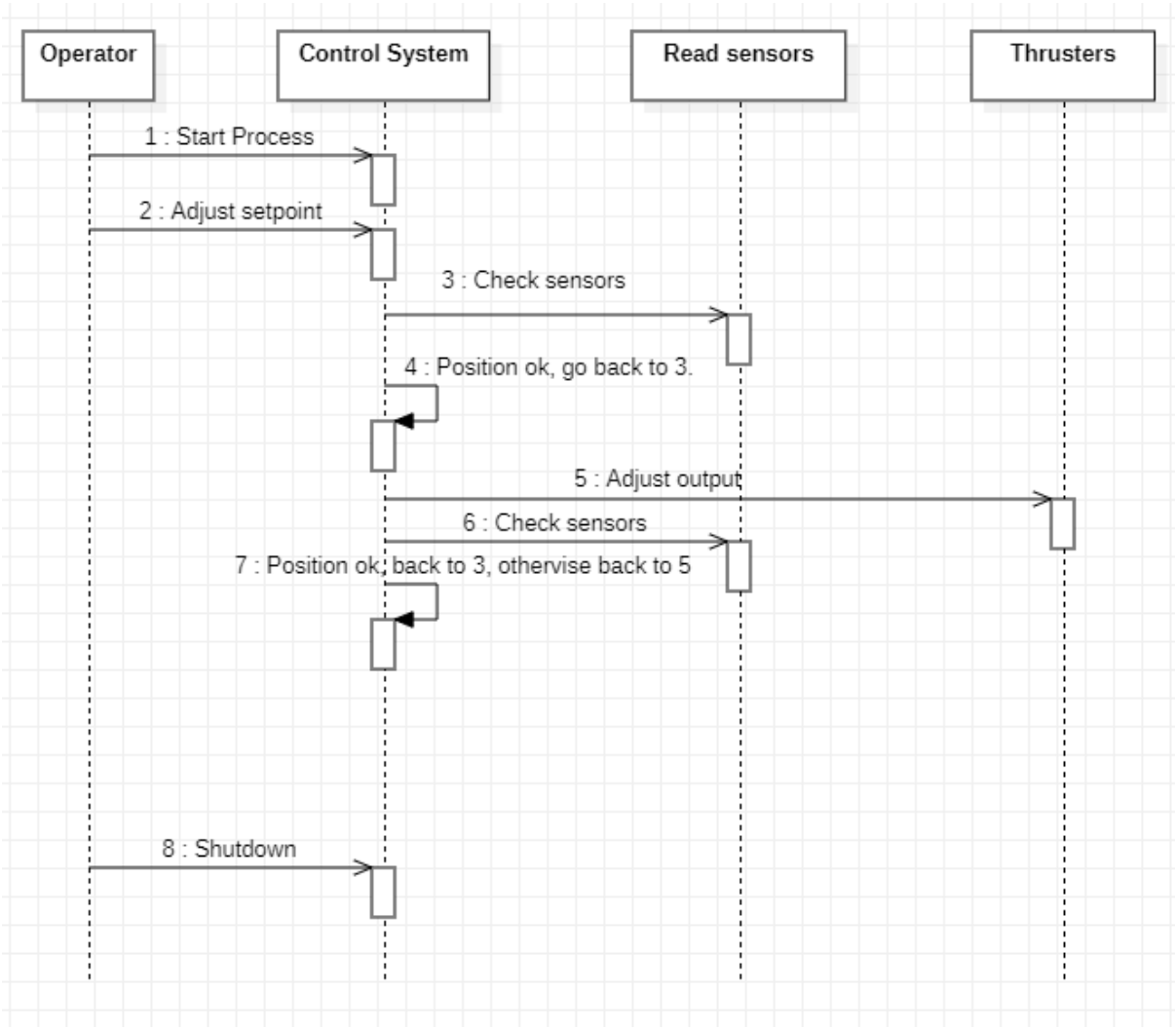


Figure 4-2 This shows a simple system sequence diagram on how the controller should operate.

4.3 Design

The design of the DP system should be easy to understand and should be simplified as much as an operator would need to run the system. Figure 4-3 shows a simplified version of the control system as a block diagram. The operator should also be familiar with the Figure 2-3 which shows the different inputs, outputs states and disturbances. The filtered value $\hat{y}$ can also be used for monitoring purposes instead of the measured value y since the filtered value should contain less noise.

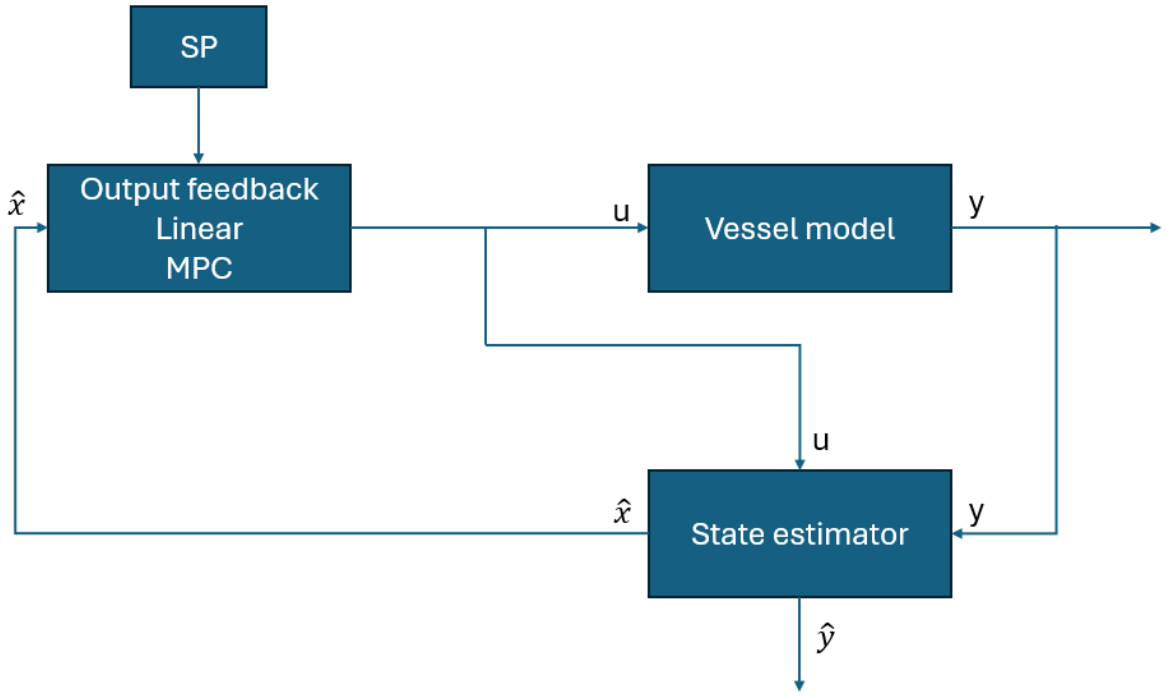


Figure 4-3 A system overview of the complete control system.

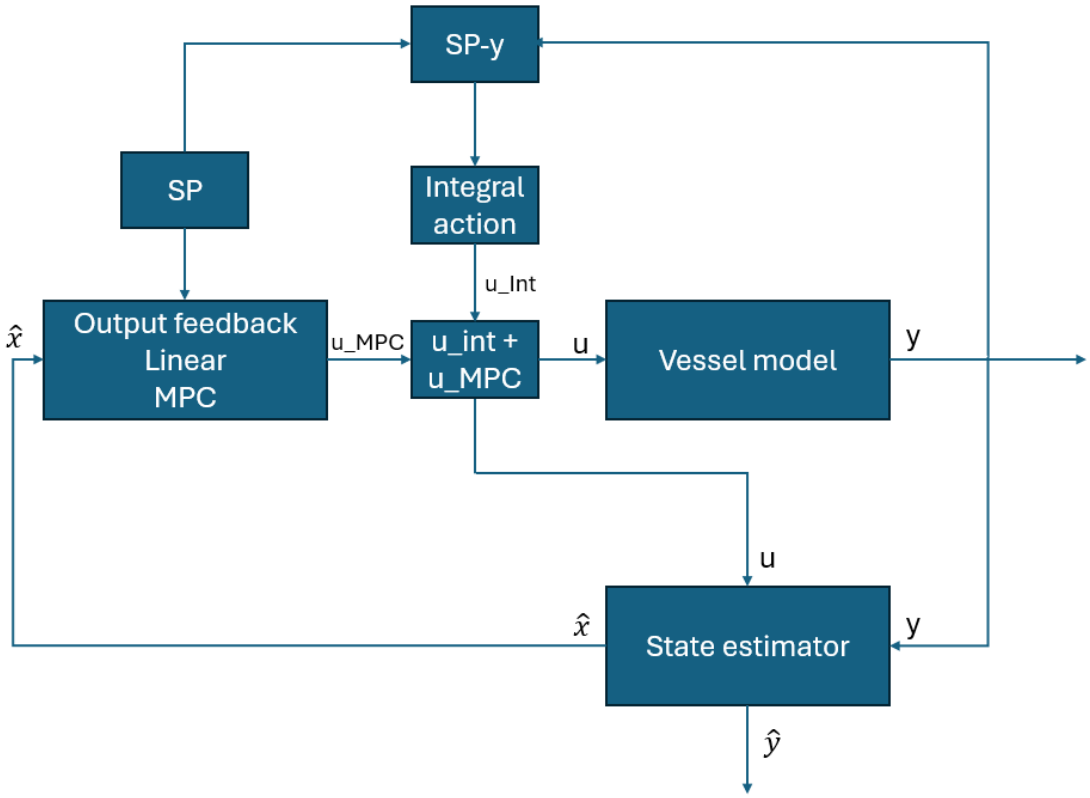


Figure 4-4 A system overview of a more advanced control system with integral action.

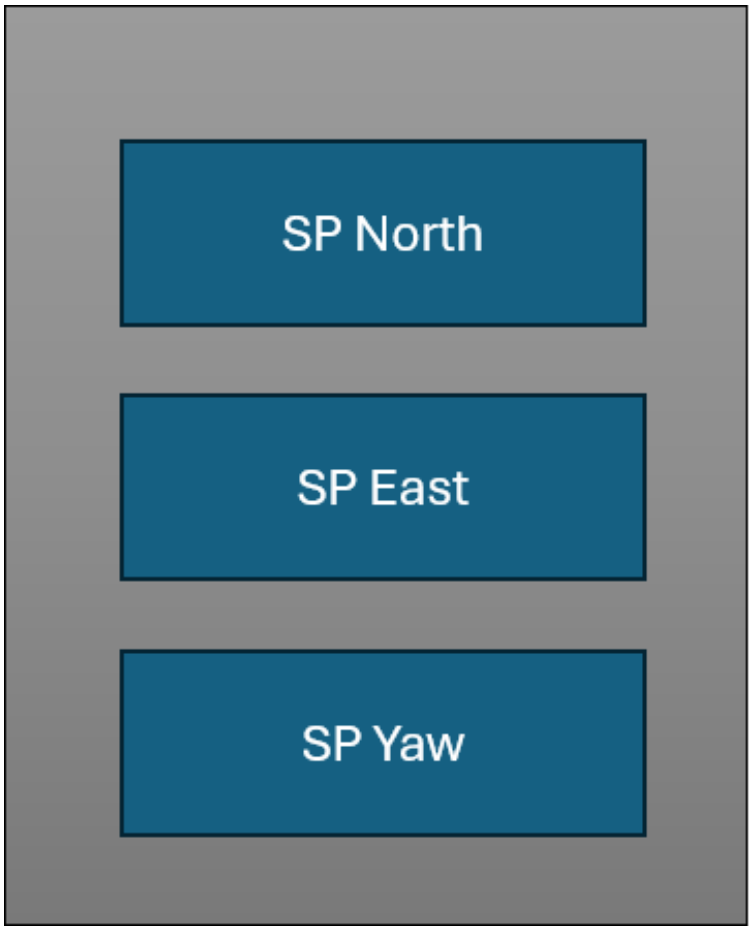


Figure 4-5 User interface for adjusting SP in North, East and Yaw in Simulink.

Figure 4-5 shows a simplified user interface which the operator can use to change the different SP in North, East, and Yaw. Due to limitation in Simulink, it is chosen to not make a more advanced user interface.

5 Results

This chapter will contain the results from this project. It will mainly follow the structure of the methods chapter unless some topics have not been pursued further.

5.1 Open loop testing

In Figure 5-1 a NE diagram is presented, based on the step response in Simulink code from Figure 3-1. These step responses generate movement of a marine vessel in both directions north and east, which can be plotted in MATLAB.

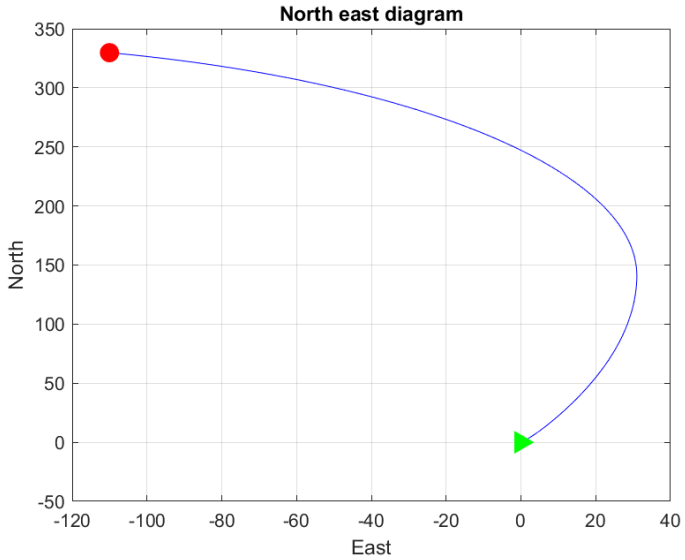


Figure 5-1 NE diagram for a step response. Green Triangle marks start, and red circle marks end. This is the first trial with steps in Surge and Sway, and a step-in yaw after 150s.

To make the NE diagram both the north position and east position must be logged to MATLABs workspace. This is done either by using a record block or a save to workspace block.

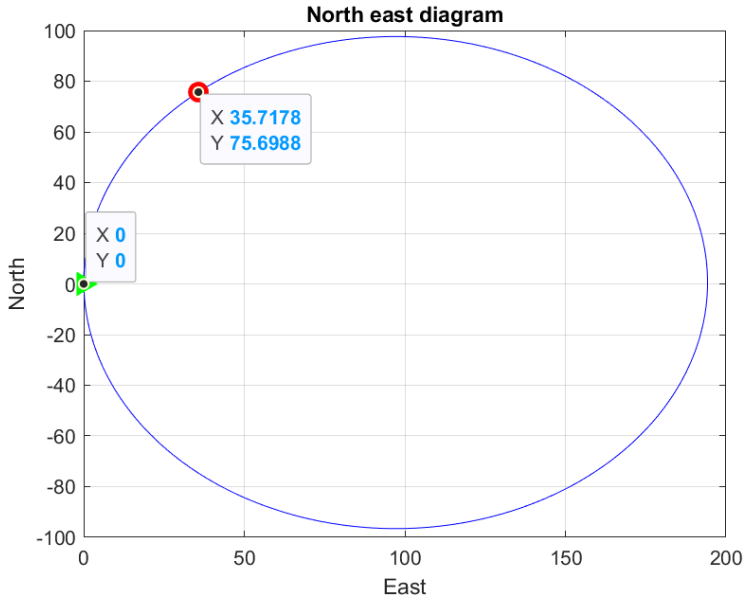


Figure 5-2 Positive control input in both yaw and surge. Green Triangle marks start, and red circle marks end.

Figure 5-2 shows the output in a NE diagram where there is given a positive control value in both yaw and surge. As expected, this is a circular movement when there are 0 disturbances, and the thrust is kept constant.

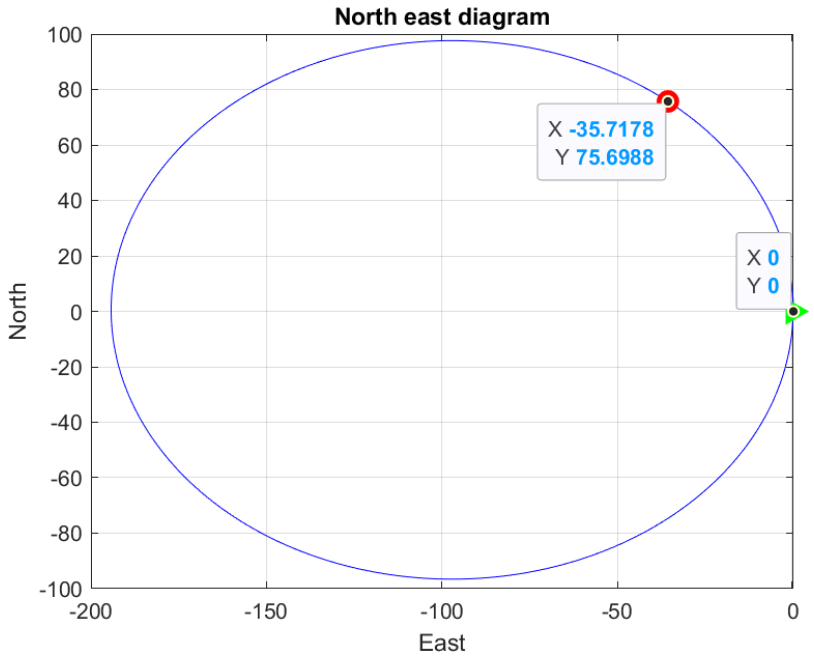


Figure 5-3 Negative control input in yaw, and positive control input in surge. Green Triangle marks start, and red circle marks end.

Figure 5-3 shows the output in a NE diagram where there is given a positive control value in surge, but a negative control value in yaw. As expected, there is a circular movement in the opposite direction as Figure 5-2.

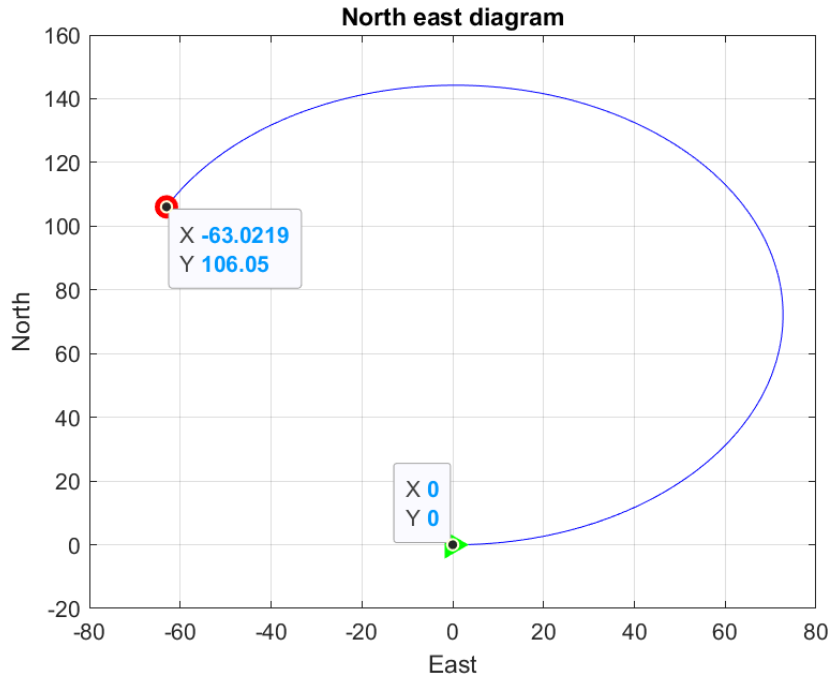


Figure 5-4 Positive control input in sway. Green Triangle marks start, and red circle marks end.

Figure 5-4 shows the response from the marine vessel with a positive control input in sway. As well as a movement to the side, the vessel also changes the yaw of the marine vessel.

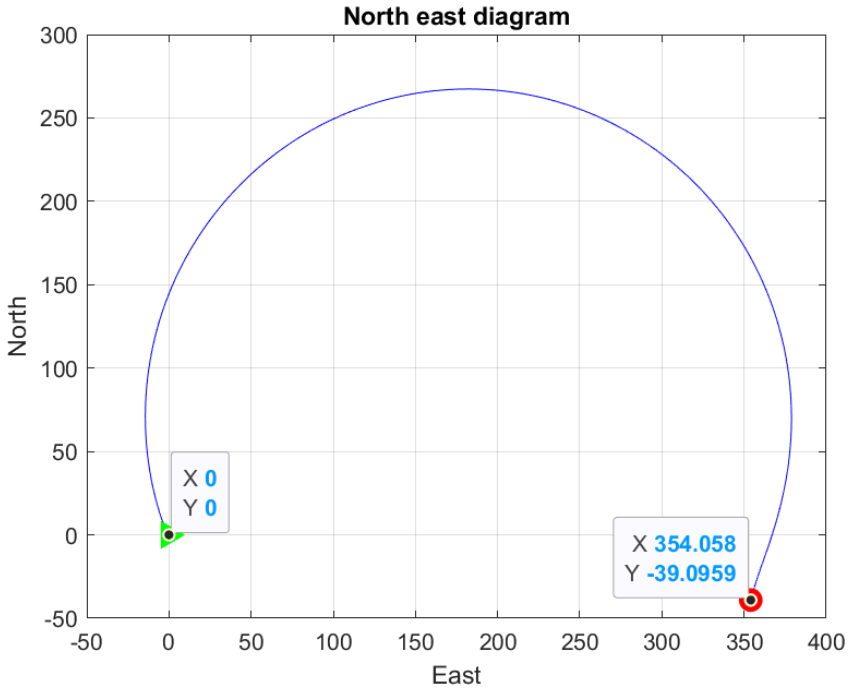


Figure 5-5 Positive control signal in surge, and negative control signal in sway. Green Triangle marks start, and red circle marks end.

Figure 5-5 shows the response from the marine vessel in a NE diagram with a positive control signal in surge, and a negative control signal in sway.

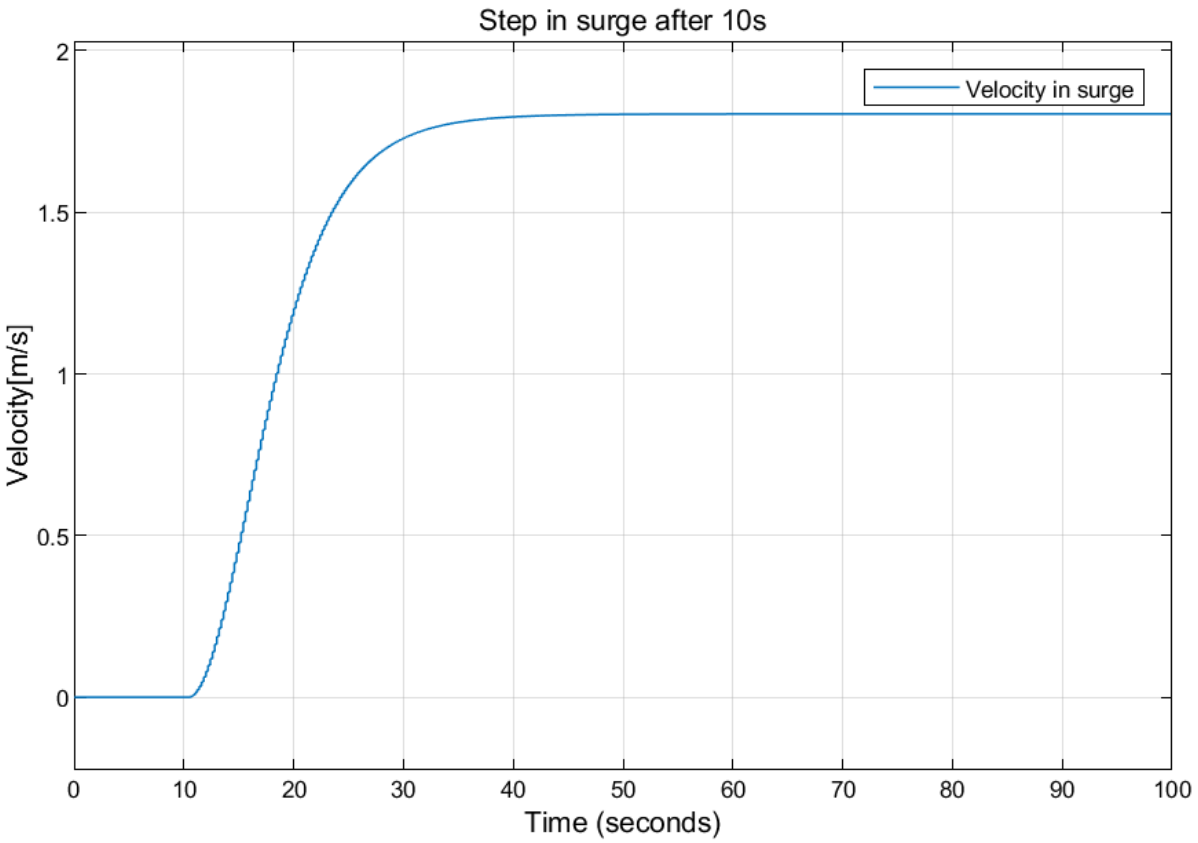


Figure 5-6 step response in surge after 10s with a force in surge with the calculated force of $265750N$ as shown in equation (3.2).

Figure 5-6 shows a step response in surge with a force of 265750N, and the Gunnerus reaches a velocity of about 1.7m/s.

5.2 System Identification

This following chapter presents the results from the system identification process of the Gunnerus marine vessel. First of in Figure 5-7 is the NE diagram where random control values in surge, sway and yaw are entered. The random control signal was multiplied by a force of 100000N to make the generated thrust more realistic so that it moves the vessel by a significant distance.

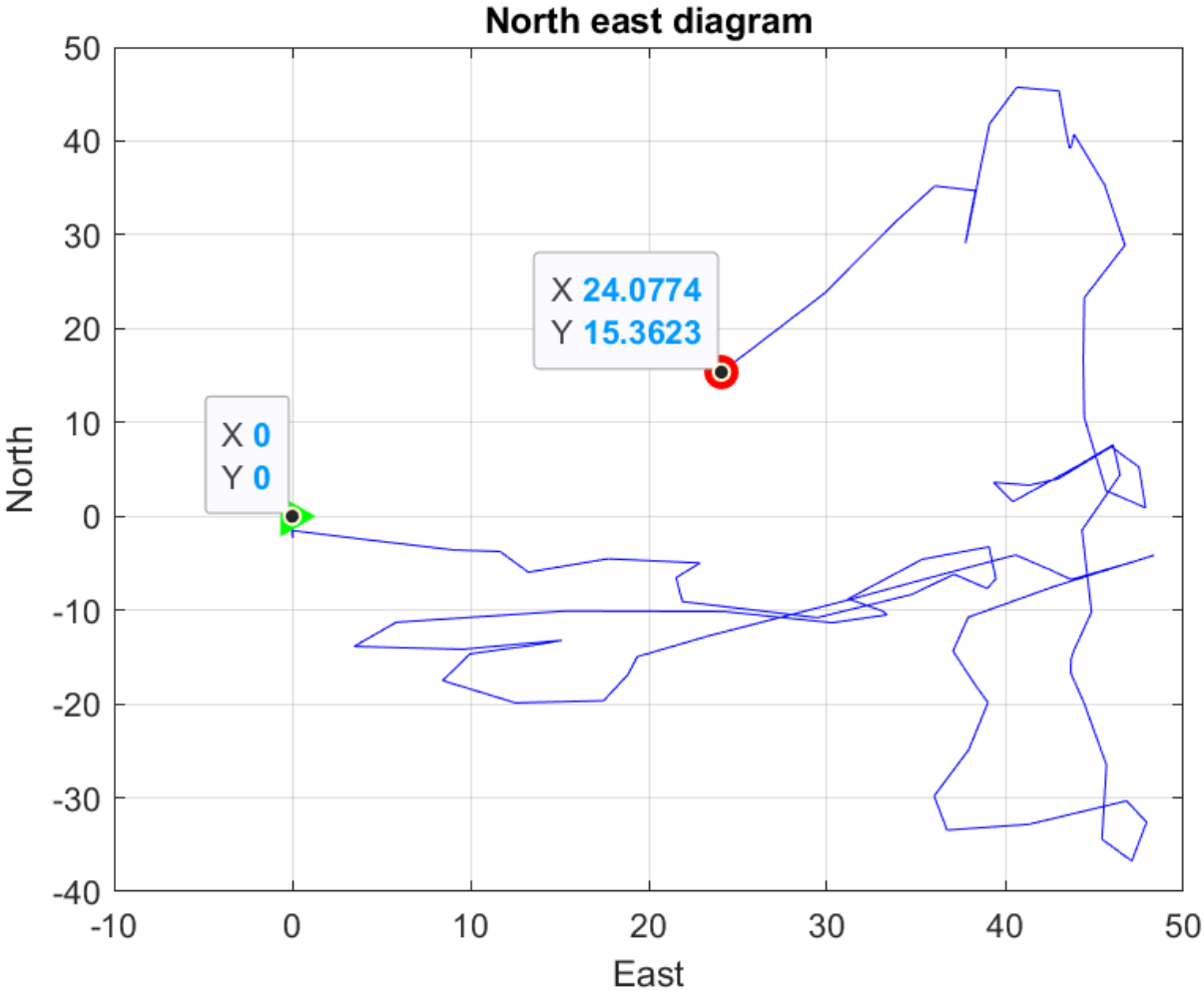


Figure 5-7 Random control values at -1 and 1 for each surge, sway, and yaw. Green Triangle marks start, and red circle marks end. The control value is multiplied with a force of 100000N to make a realistic force to move the vessel.

Figure 5-8, Figure 5-9, and Figure 5-10 shows an example of the generated square pulses and the thrust input they generate in each surge, sway, and yaw.

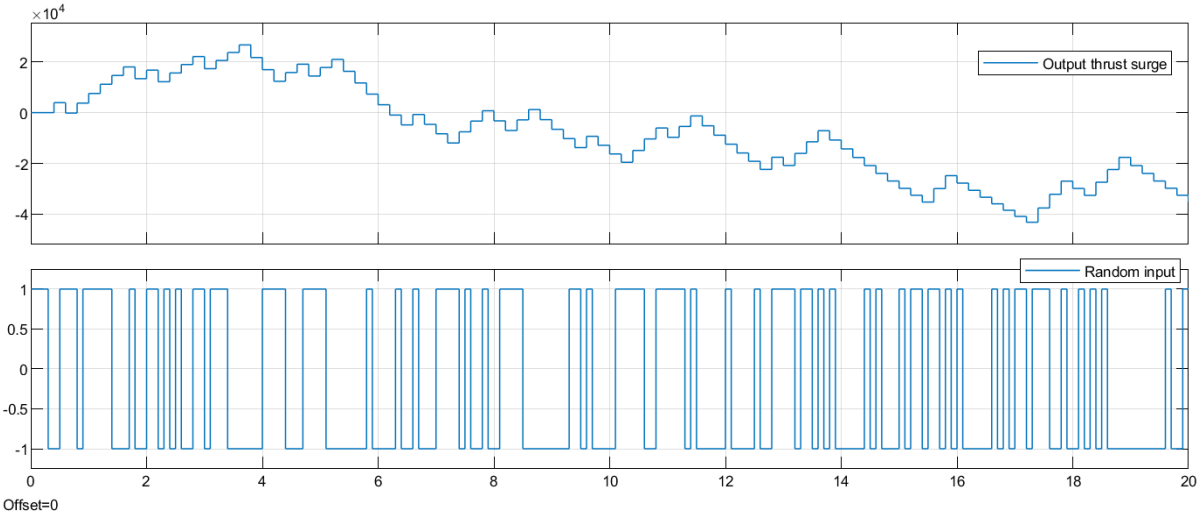


Figure 5-8 Example of a random input signal between 1 and -1 and the thrust it generates in surge direction.

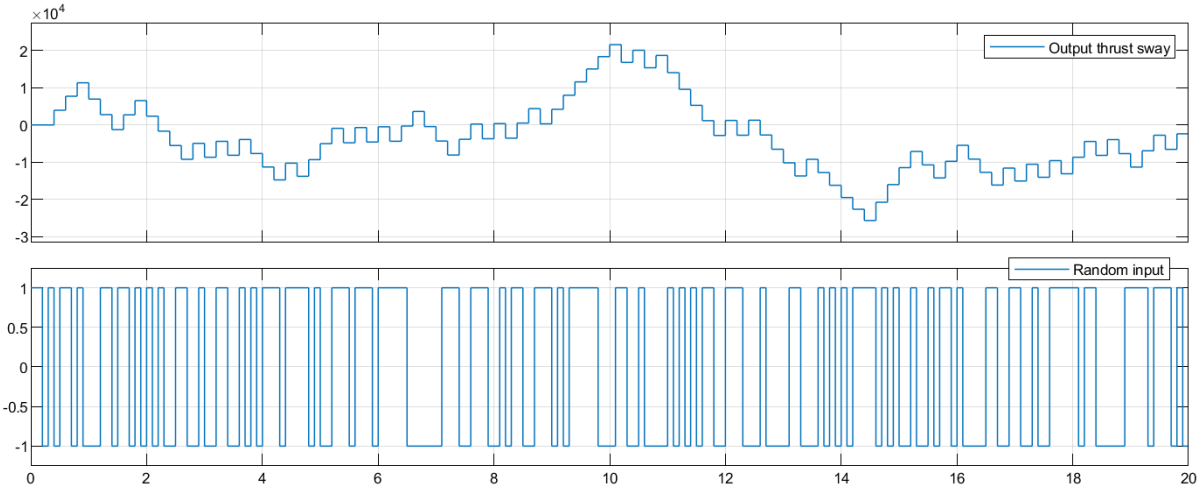


Figure 5-9 Example of a random input signal between 1 and -1 and the thrust it generates in sway direction.

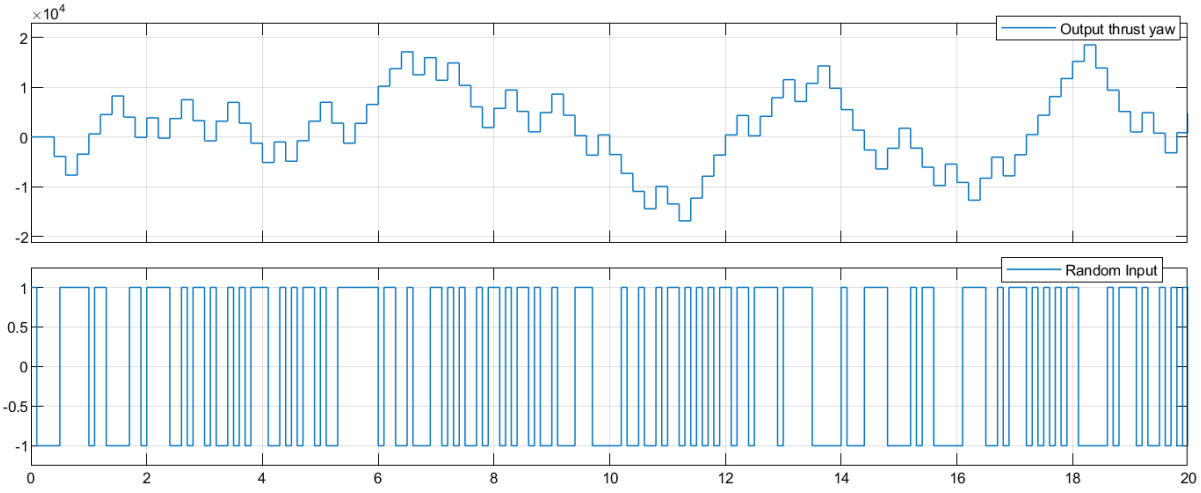


Figure 5-10 Example of a random input signal between 1 and -1 and the thrust it generates in yaw direction.

5.2.1 System matrices.

The matrices that represent the system well are given below. These matrices were found using the D-SR function as depicted in equation (3.3). The Kalman filter gain is calculated using equation (3.4) and is given as the matrix K.

$$A = \begin{bmatrix} 1.0034 & 0.0207 & 0.4737 & 0.0463 & -0.2364 & 2.3069 \\ -0.001 & 0.9997 & -0.054 & 0.1380 & 0.9540 & -13.3772 \\ 0 & 0 & 0.9424 & 0.6183 & -0.2177 & 2.1255 \\ 0 & 0 & 0.1676 & -0.8000 & 0.4213 & -5.1737 \\ 0 & -0.0002 & -0.0060 & 0.1014 & -0.1223 & -12.4611 \\ 0 & 0 & 0 & -0.0004 & -0.0028 & -0.5637 \end{bmatrix} \quad (5.1)$$

$$B = 1.0e-08 \cdot \begin{bmatrix} -0.8484 & -0.3258 & -0.1868 \\ 0.4781 & -0.1277 & -0.2667 \\ -0.0948 & -0.0255 & -0.0254 \\ 0.2341 & 0.0767 & 0.1066 \\ -0.0197 & -0.0031 & -0.1006 \\ -0.0005 & -0.0006 & 0.0006 \end{bmatrix} \quad (5.2)$$

$$C = \begin{bmatrix} 0.6830 & -0.1854 & -0.2271 & 0.6690 & 0.0090 & 0.0006 \\ 0.1784 & 0.6819 & -0.0555 & -0.0024 & -0.7072 & 0.0085 \\ -0.0048 & -0.0269 & -0.6671 & -0.2302 & 0.0345 & 0.7071 \end{bmatrix} \quad (5.3)$$

$$D = 1.0e-08 \cdot \begin{bmatrix} -0.3458 & -0.1652 & -0.0717 \\ -0.0376 & -0.0129 & -0.1373 \\ 0.0011 & 0.0020 & -0.0003 \end{bmatrix} \quad (5.4)$$

$$CF = \begin{bmatrix} 0.0262 & 0.0068 & 0.0001 \\ -0.0201 & 0.0256 & 0.0002 \\ 0.0036 & -0.0004 & -0.0004 \\ -0.0095 & -0.0001 & -0.0001 \\ -0.0044 & 0.0088 & 0.0002 \\ 0 & 0 & -0.0001 \end{bmatrix} \quad (5.5)$$

$$F = 1.0e-03 \cdot \begin{bmatrix} 0.7813 & 0 & 0 \\ -0.1125 & 0.6927 & 0 \\ 0.0020 & -0.0019 & 0.0111 \end{bmatrix} \quad (5.6)$$

$$x0 = \begin{bmatrix} -0.0031 \\ 0.0010 \\ -0.0005 \\ 0.0014 \\ -0.0001 \\ 0 \end{bmatrix} \tag{5.7}$$

$$K = \begin{bmatrix} 34.8690 & 9.8057 & 11.2373 \\ -20.4980 & 36.9942 & 21.6928 \\ 4.6280 & -0.6726 & -33.1105 \\ -12.2170 & -0.1861 & -5.2135 \\ -3.8993 & 12.7559 & 16.9157 \\ 0.0335 & -0.0487 & -11.0087 \end{bmatrix} \tag{5.8}$$

5.2.2 System analysis.

The eigenvalues were found using the eig.m function in MATLAB.

$$eig(A) = \begin{bmatrix} 1.0016 + 0.0045i \\ 1.0016 - 0.0045i \\ 0.9995 + 0.0000i \\ -0.0002 + 0.0000i \\ -0.9134 + 0.0000i \\ -0.6296 + 0.0000i \end{bmatrix} \tag{5.9}$$

As equation (5.9) shows the system is unstable. The reason for this is the eigenvalues that are bigger than 1.

Using the ctrb.m function it is possible to calculate the rank of the controllability matrix. The method in MATLAB is shown in Figure 5-11, and the rank was found to be 6. This means that the system is controllable.

```
>> rank(ctrb(A, B))

ans =

     6
```

Figure 5-11 Using MATLAB to find the rank of the controllability matrix.

The same method can be done for the observability matrix using the obsv.m function in MATLAB. As shown in Figure 5-12 the rank of the observability matrix is equal to 6. This means that the system is observable since the rank of the system is 6.

```
>> rank(obsv(A, C))

ans =

     6
```

Figure 5-12 Using MATLAB to find the rank of the observability matrix. Note, MATLAB notation is C and not D as in equation

(3.6).

5.3 State estimator and Kalman filter.

In this following chapter different experiments are carried out to test the performance of the Kalman filter in the state estimator. The model settings are shown in Figure 5-13. Notice that in this chapter there was an error in the NE script in MATLAB that switched the axes of north and east.

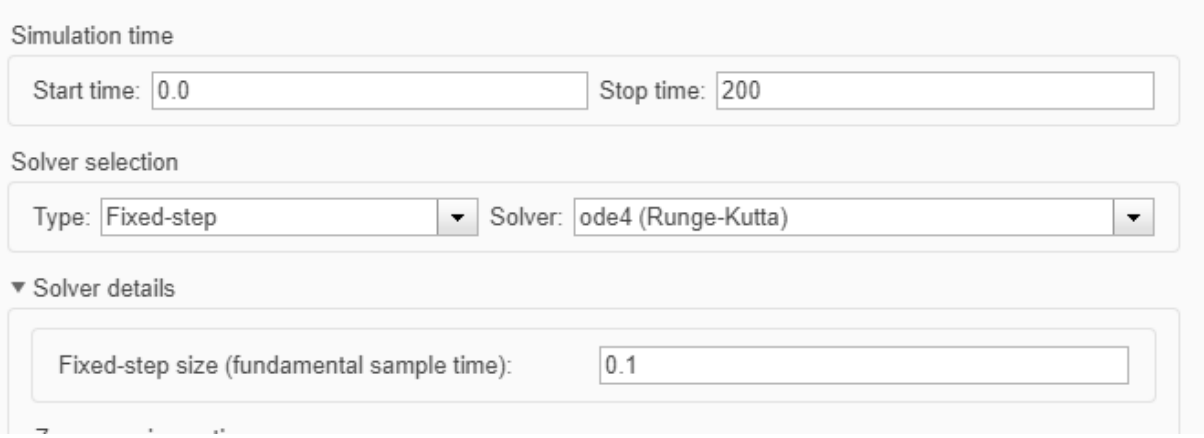


Figure 5-13 Model settings for simulating in Simulink.

The first experiment is carried out with a limited time span of 20s and a force of 50000N randomly inserted in surge, sway, and yaw. The force can also be negative. The results are shown in Figure 5-14, Figure 5-15, and Figure 5-16 below.

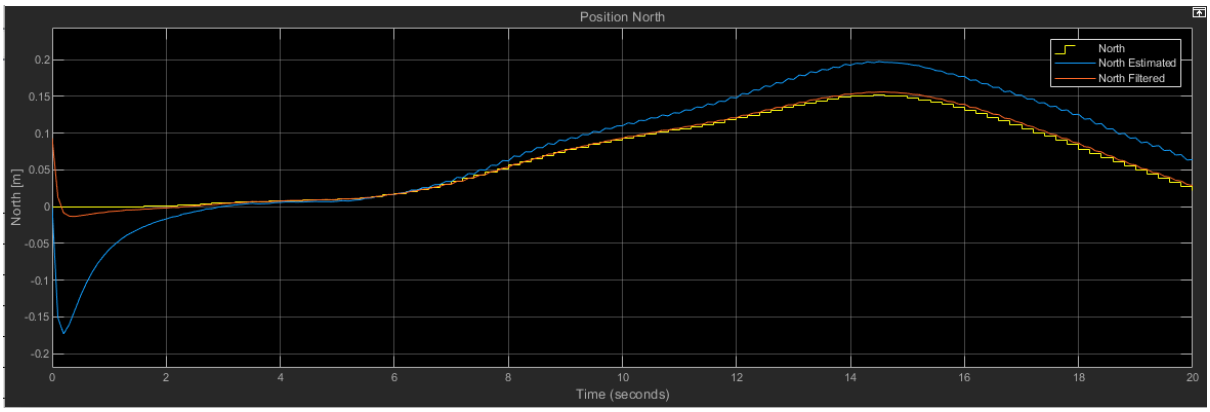


Figure 5-14 State estimator test with random input values, here north is shown. Here the real process is marked in yellow, the estimated value is marked in blue, and the filtered value marked in orange.

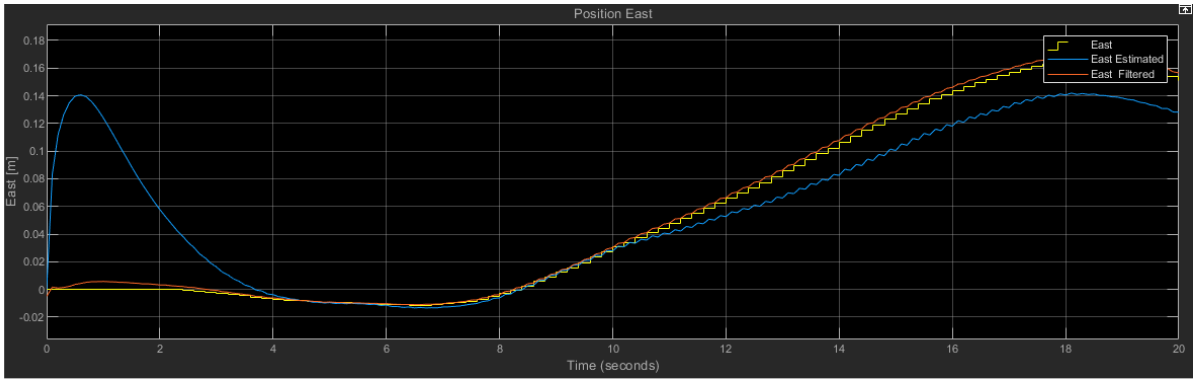


Figure 5-15 State estimator test with random input values, here east is shown. The real process is marked in yellow, the estimated value is marked in blue, and the filtered value marked in orange.

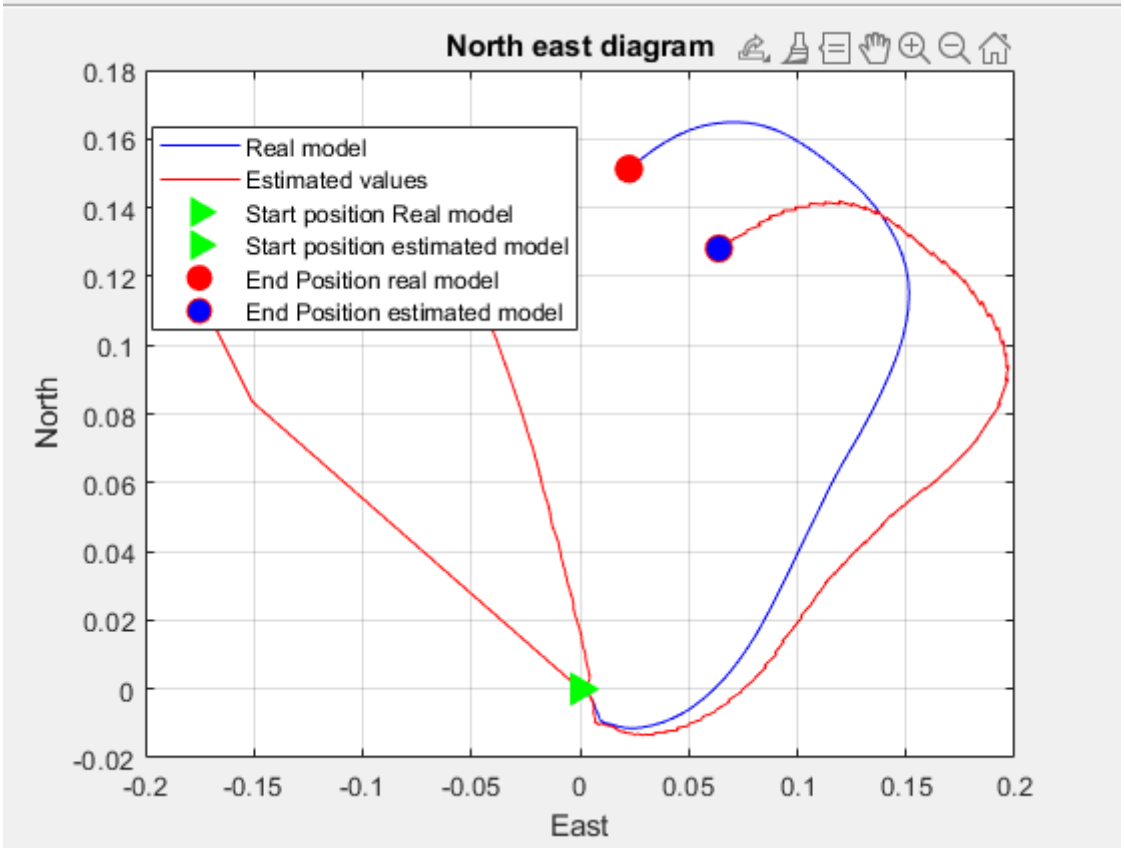


Figure 5-16 NE diagram with random control input values. Notice the state estimator having an extra loop to the left before it follows the real model.

The second test was with a time span of 200s and a random input force of 50000N in surge, sway, and yaw. Also here, the input force could be negative. Figure 5-17, Figure 5-18, Figure 5-19, and Figure 5-20 shows the result from this experiment.

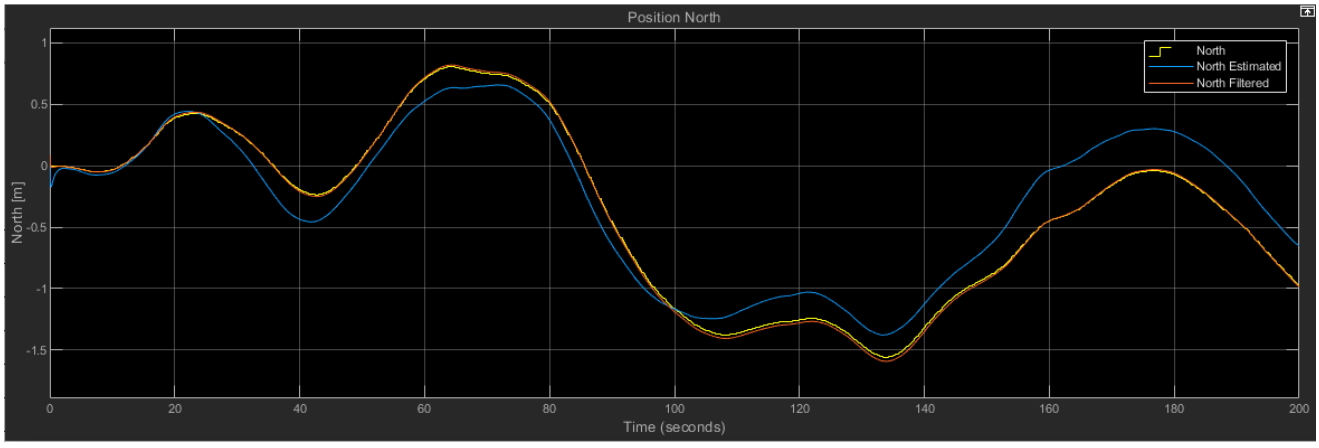


Figure 5-17 The yellow line is north; the blue line is the estimated value, and the orange line is the filtered value for north. The timespan is 200s and the input force in surge, sway, and yaw is 50000.

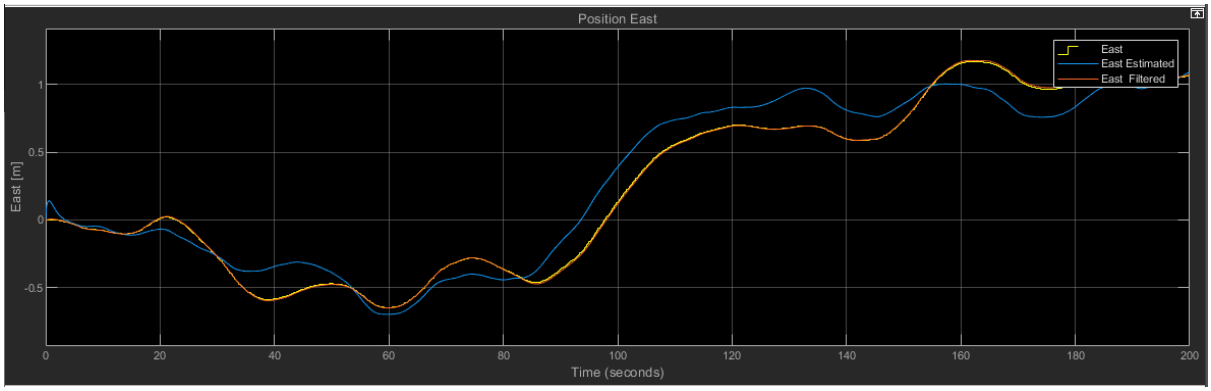


Figure 5-18 The yellow line is east; the blue line is the estimated east value, and the orange line is the filtered value for east. Timespan is 200s.

.

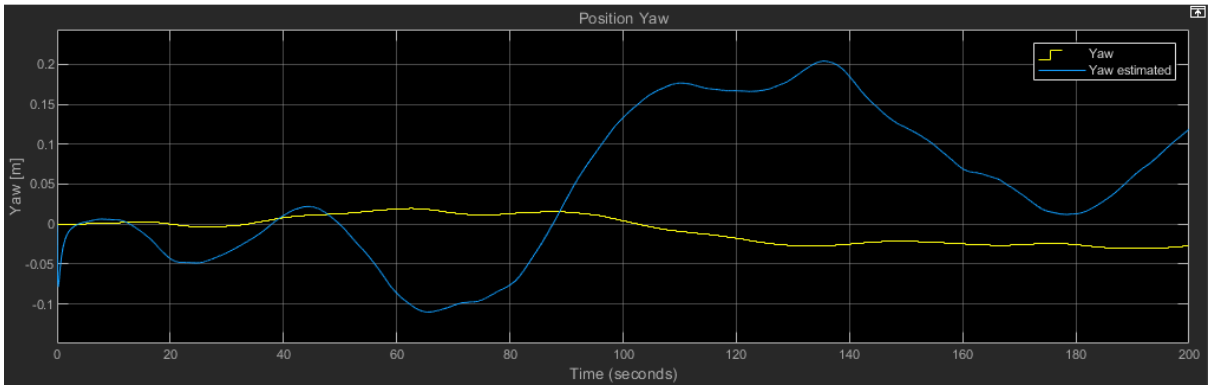


Figure 5-19 The yellow line is measured yaw, and the blue line is the estimated yaw. Timespan is 200s.

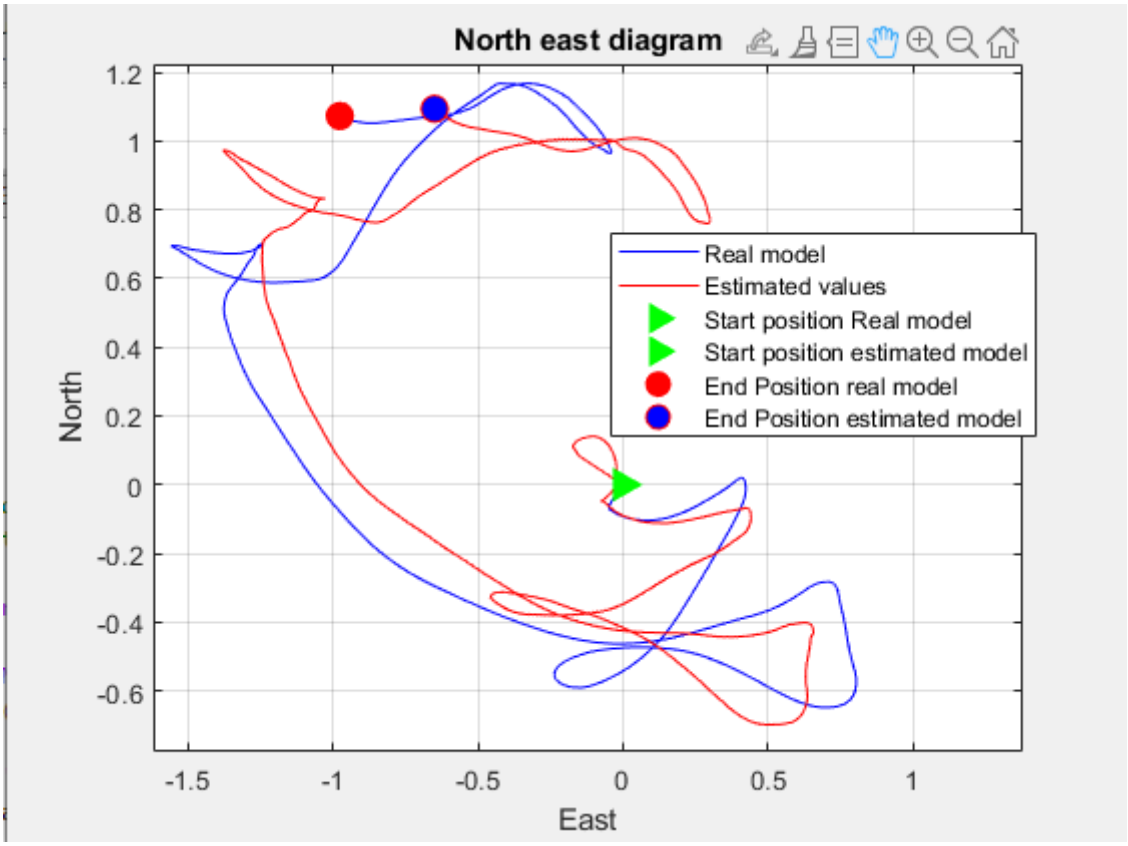


Figure 5-20 NE diagram with random values. Notice that the extra loop the state estimator makes is a lot less significant than in Figure 5-16.

The third test was with a time span of 200s and a random input force of 100000N in surge, sway, and yaw. Also here, the input force could be negative. Figure 5-21, Figure 5-22, Figure 5-23, and Figure 5-24 shows the result from this experiment.

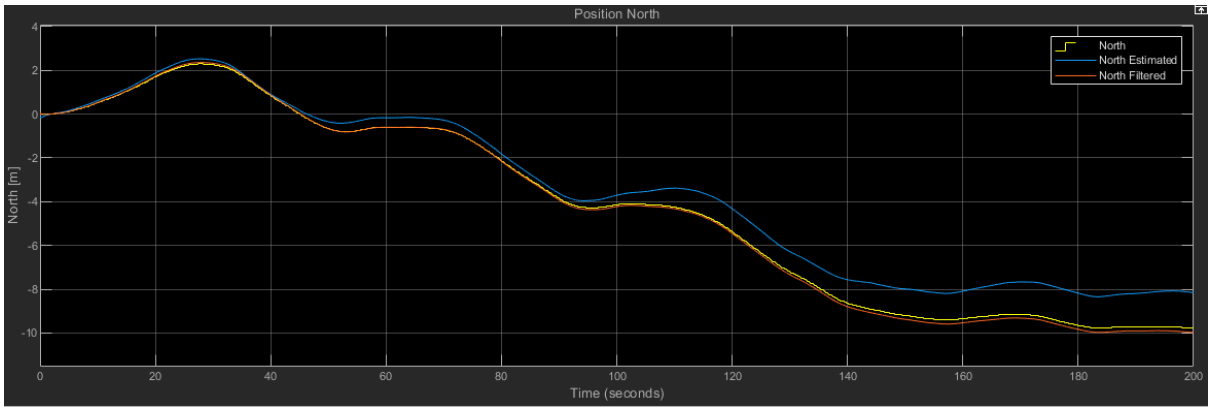


Figure 5-21 The yellow line is north; the blue line is the estimated value, and the orange line is the filtered value for north. The time span is 200s and the input force in surge, sway, and yaw is 100000N.

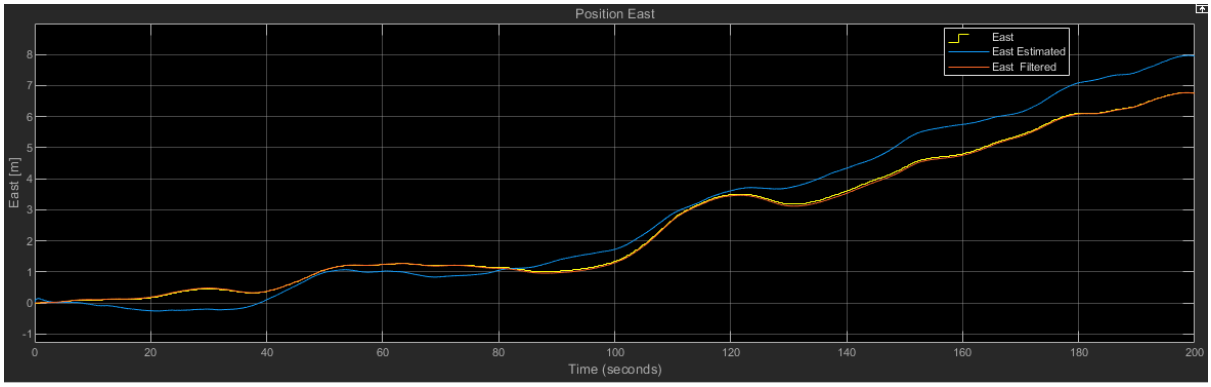


Figure 5-22 The yellow line is east; the blue line is the estimated value, and the orange line is the filtered value for east. The time span is 200s and the input force in surge, sway, and yaw is 100000N.

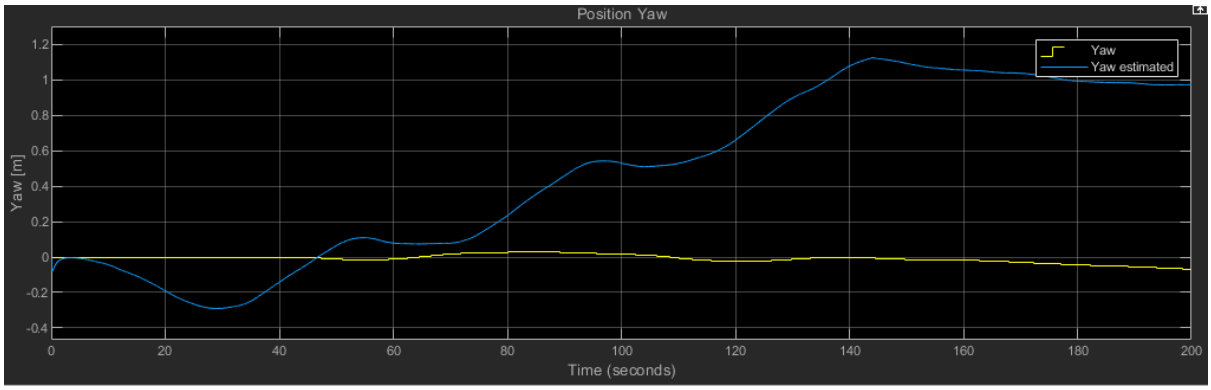


Figure 5-23 The yellow line is yaw; the blue line is the estimated value. The time span is 200s and the input force in surge, sway, and yaw is 100000N.

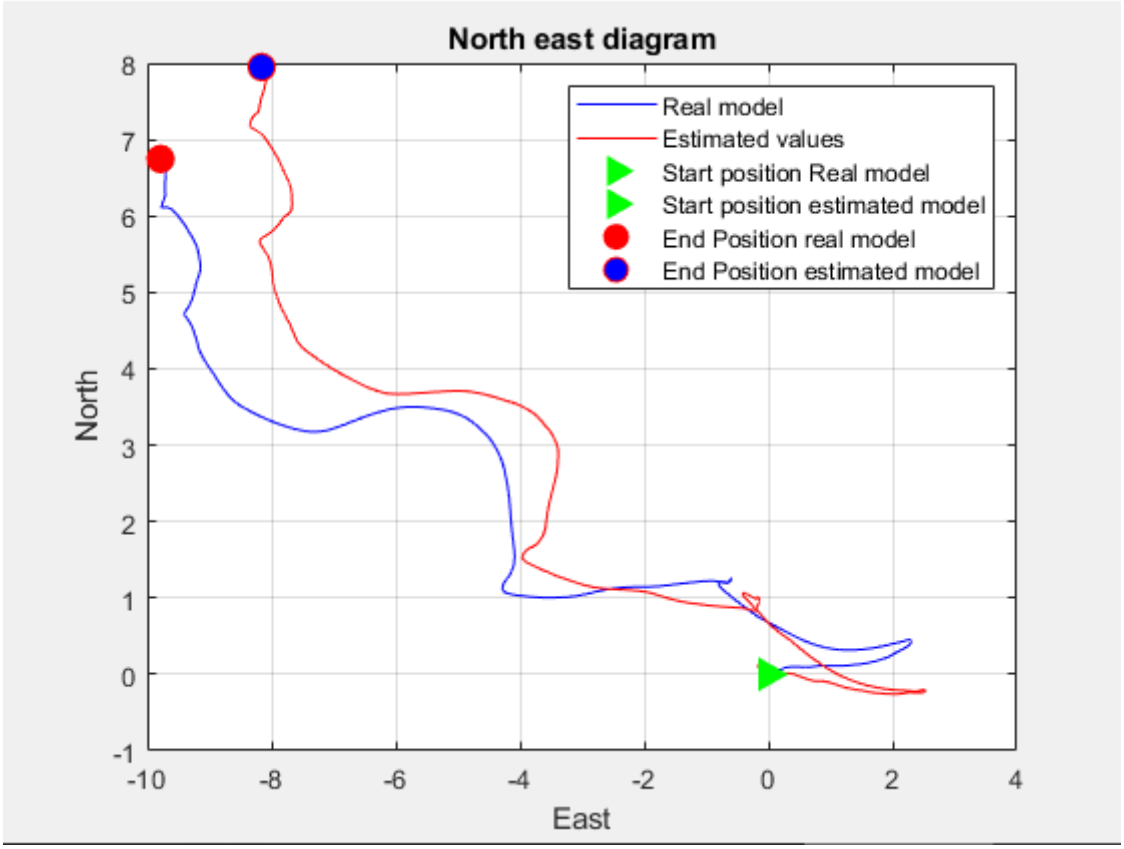


Figure 5-24 NE diagram with random values. Notice that the extra loop the state estimator makes in the start is now negligible. But the deviation in end point is a little bigger than in Figure 5-20.

5.4 Control of the marine vessel

This chapter is divided into two parts, one part with an MPC controller with feedback from a state estimator, and one expanded version with integral action.

5.4.1 MPC Controller

The controller type for the marine vessel was selected to be an MPC controller. There were done several tests on how the controller behaved. Before the test was stopped, the control values in surge, sway and yaw were checked so the simulation wasn't stopped in the middle of a control action.

The first test was focused on the north direction. The settings for the weighting matrices Q and P are shown in Figure 5-25.

```
N = 20;                          %PRediction horizon length
Q = diag([2e4 5e4 1e4 ]);        %Weighting matrix for the error
%%Q = diag([1 1 1 ]);          %Weighting matrix for the error
%P = 1e-3;                       % Weighting matrix for the input
P = diag([2e-6 2e-6 1e-6]);
```

Figure 5-25 Simulink settings for testing the controller in the north direction. These settings are set in the QP problem formulation script block as shown in Figure 3-14.

The results from this simulation is shown in Figure 5-26,Figure 5-27,Figure 5-28, and Figure 5-29. The real model never reaches SP exactly but has a constant deviation. This might be because linear MPC is used on a nonlinear model.

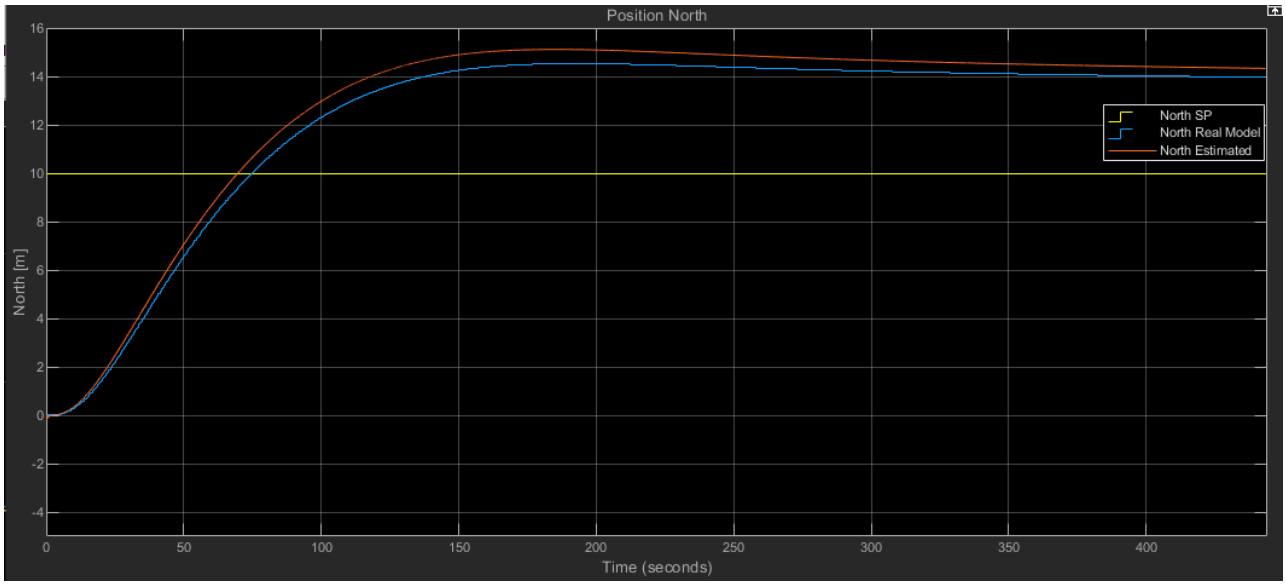


Figure 5-26 Here is the result of a SP= 10 in North direction. As shown the model reaches a steady state after about 400s. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

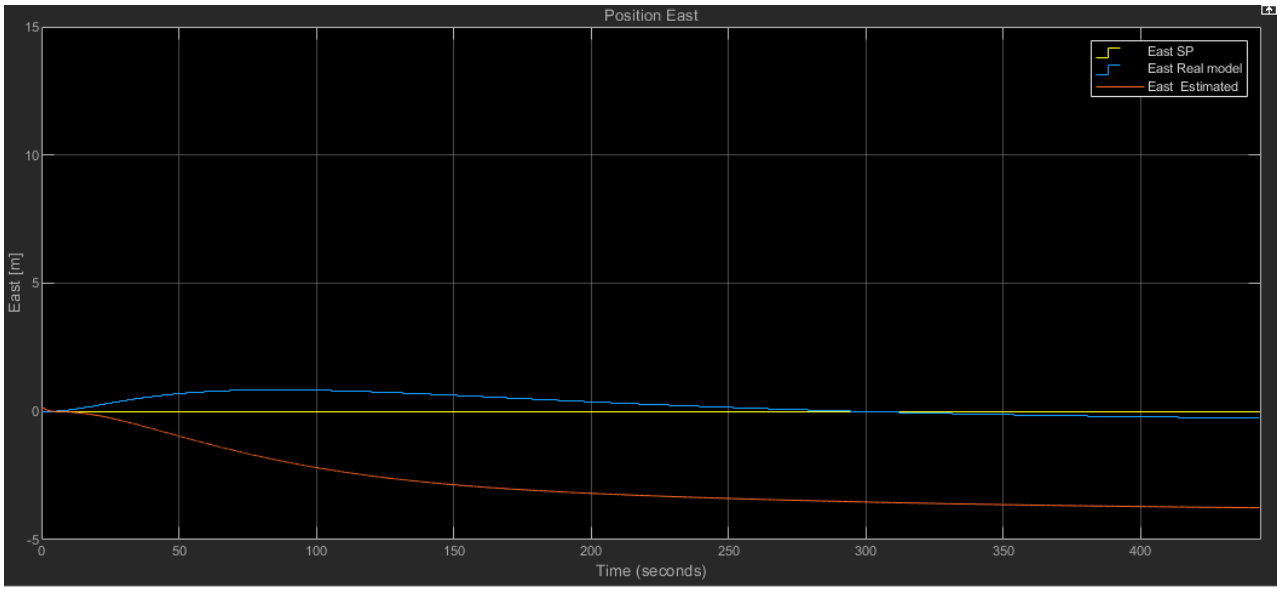


Figure 5-27 Here the position in East is shown as the result of a SP=10 change in North direction. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

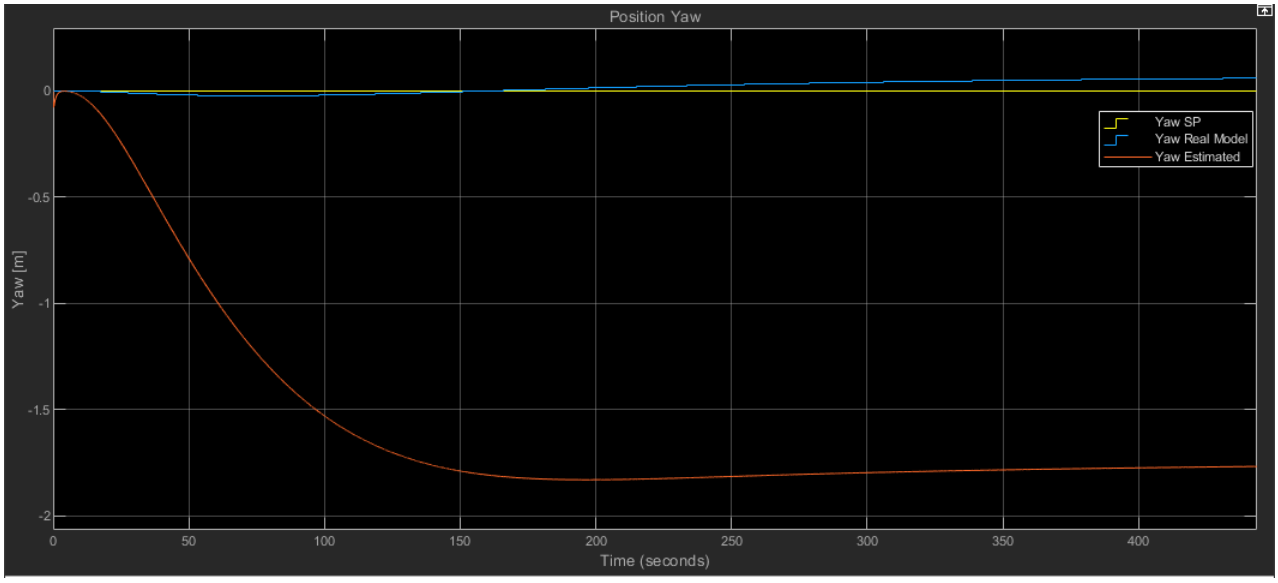


Figure 5-28 The estimated values for yaw don’t seem quite right.

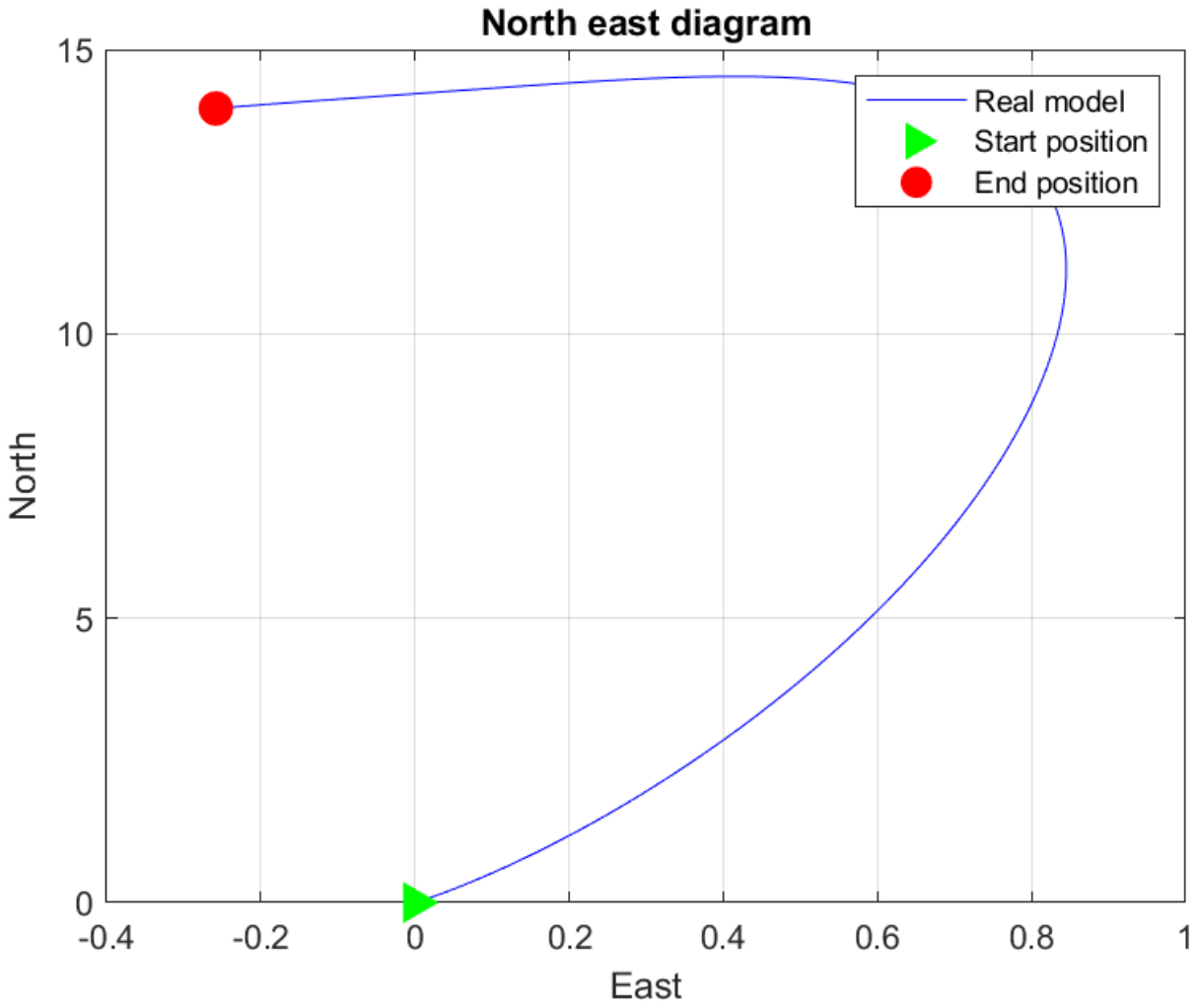


Figure 5-29 NE diagram for a SP=10 change in north direction. Here it is shown that the Gunnerus ship is making a turn even though it is not necessary to reach the end position.

The second test was done in the east direction. The settings are shown in Figure 5-30 for the weighting matrices Q and P.

```
N = 20;                          %PRediction horizon length
Q = diag([2e4 5e4 1e4 ]);        %Weighting matrix for the error
%%Q = diag([1 1 1 ]);          %Weighting matrix for the error
%P = 1e-3;                       % Weighting matrix for the input
P = diag([2e-6 2e-6 1e-6]);
```

Figure 5-30 weighting settings for Q and P for simulation in east direction. Prediction horizon is 20.

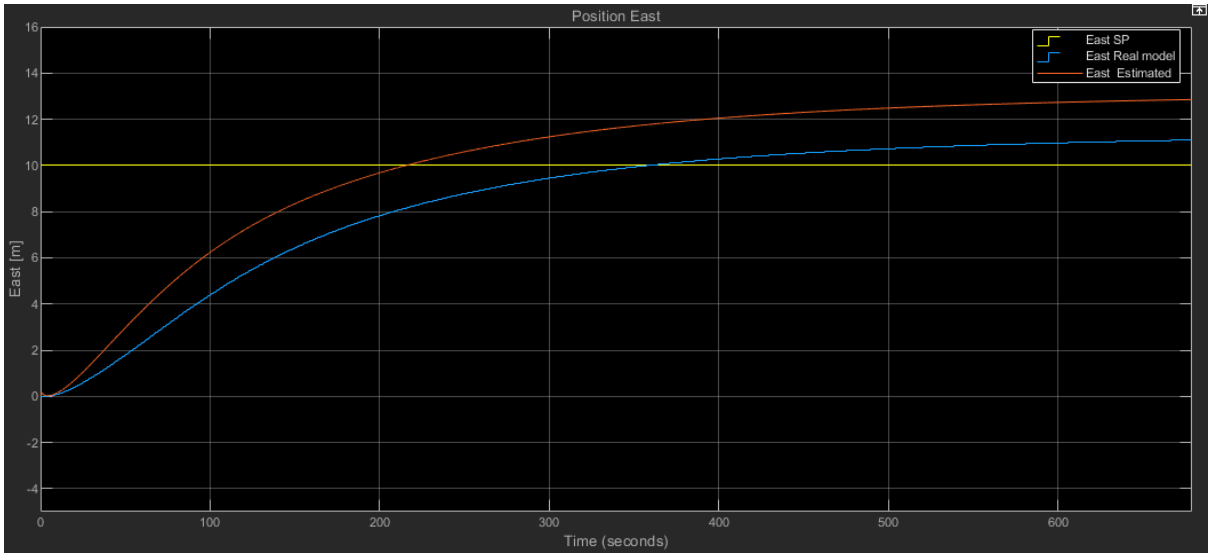


Figure 5-31 A SP=10 change in the east direction. Notice that the deviation from SP is smaller than the experiment in north direction. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

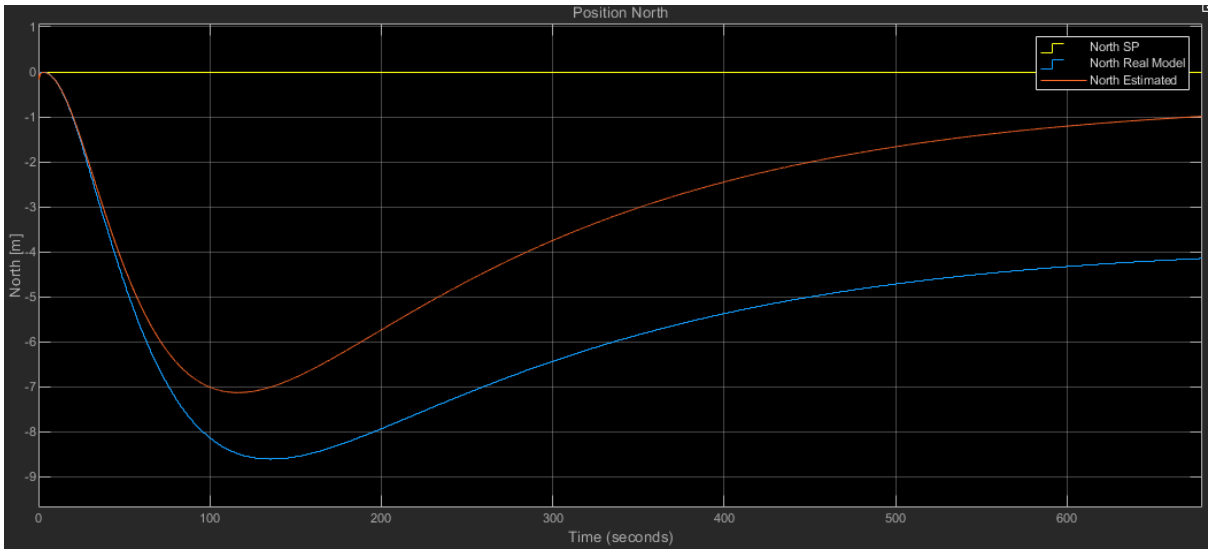


Figure 5-32 Deviation in north when running a simulation for SP=10 in east direction and SP=0 in north direction. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

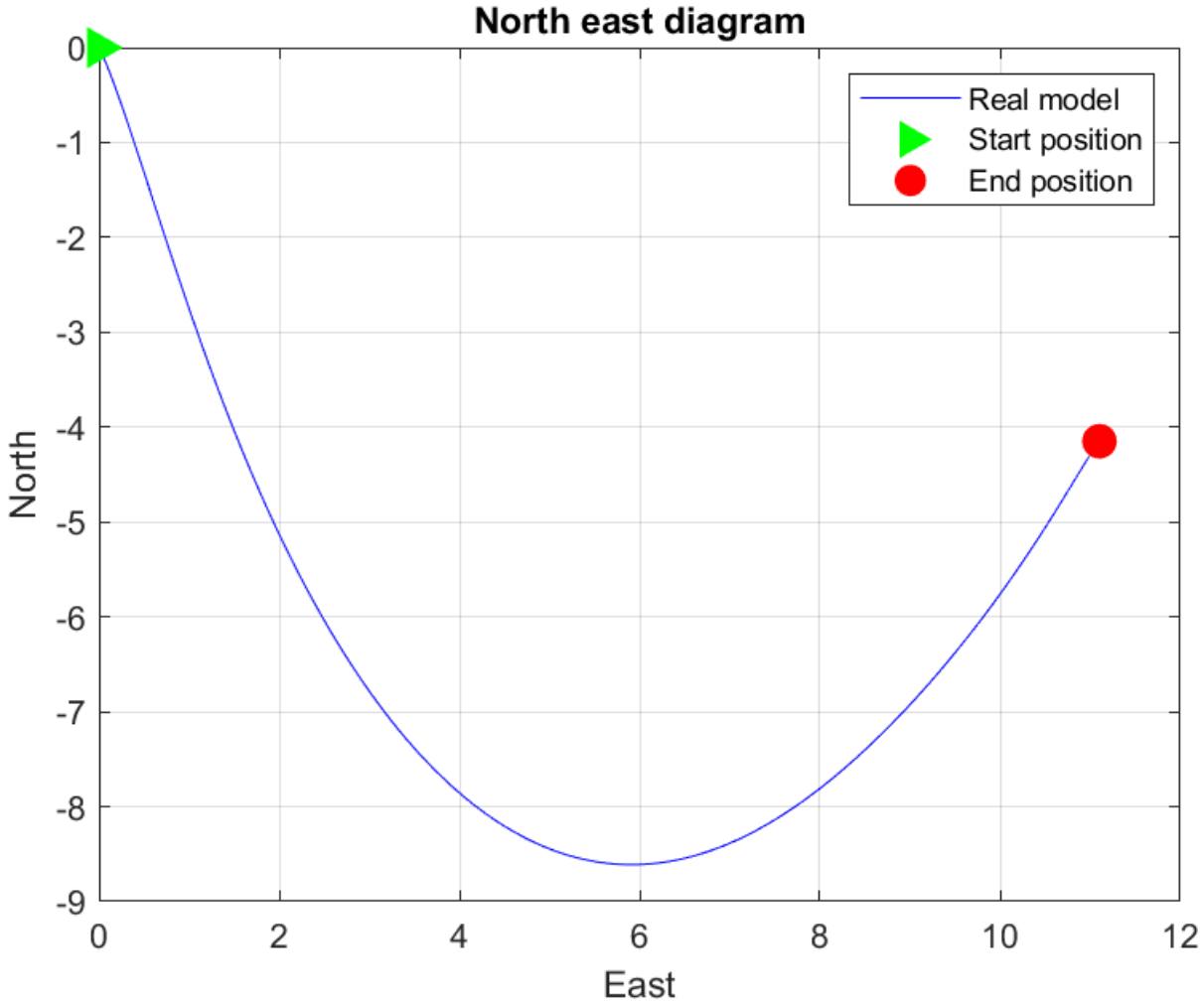


Figure 5-33 NE diagram with SP=10 in east direction. SP=0 in north and SP=0 in yaw.

The third test was with a SP=10 in north direction, and SP= 10 in east direction. The settings for the weights are shown in Figure 5-34.

```
7
8   N = 20;                          %PRediction horizon length
9   Q = diag([3e4 5e4 1e4 ]);        %Weighting matrix for the error
10  %P = 1e-3;                       % Weighting matrix for the input
11  P = diag([2e-6 2e-6 1e-6]);
12
13
```

Figure 5-34 Weight settings for Q and P. SP=10 in North and SP=10 in East.

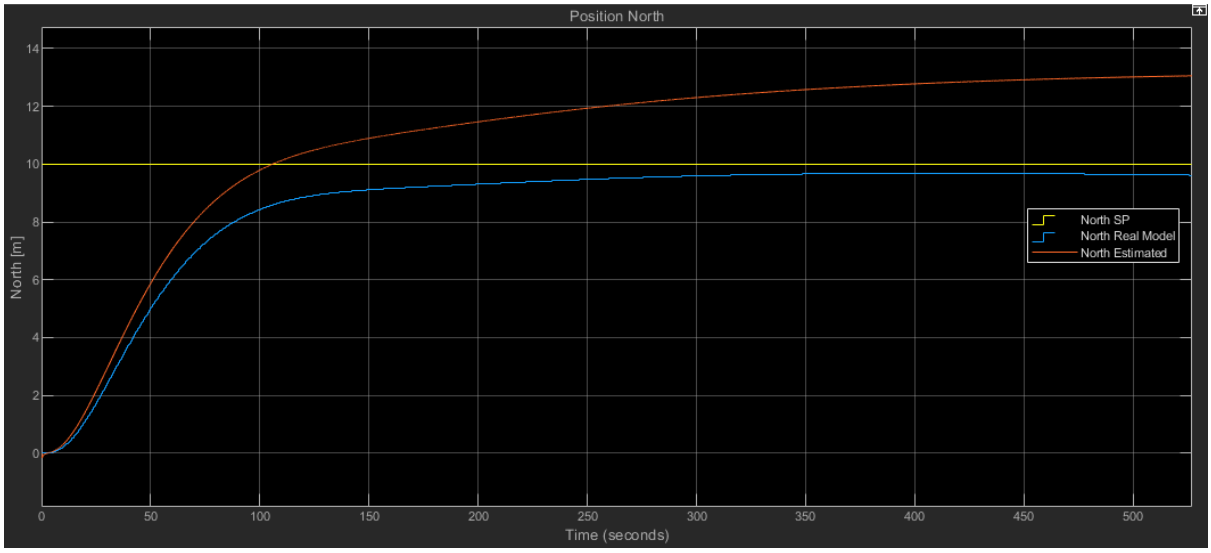


Figure 5-35 Simulation with SP=10 in north direction and SP=10 in east direction. Here are the results for the north direction. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

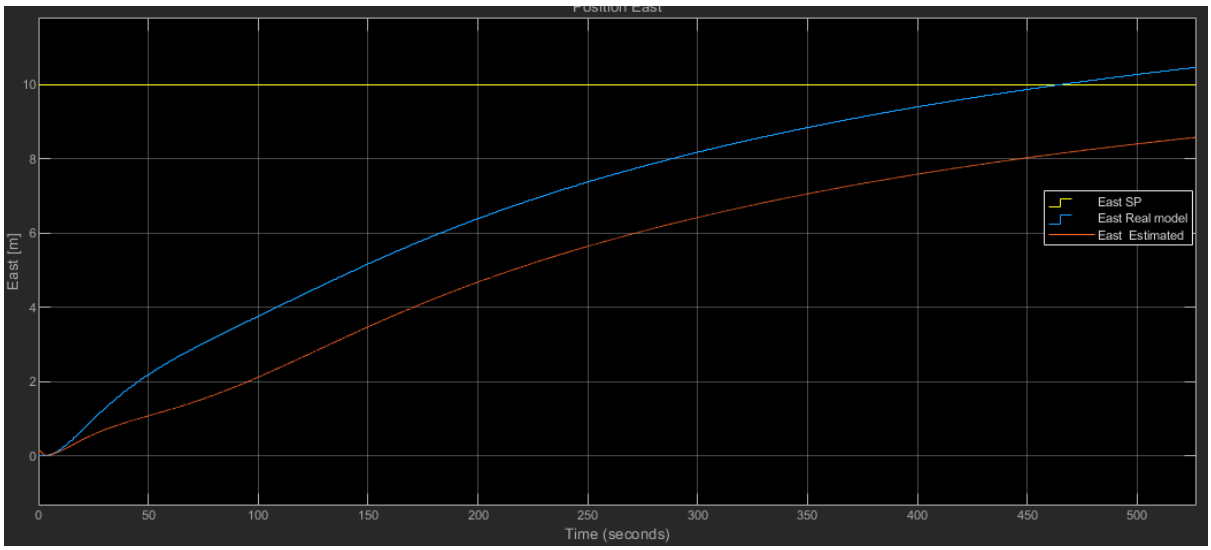


Figure 5-36 Simulation with SP=10 in north direction and SP=10 in east direction. Here are the results for east direction. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

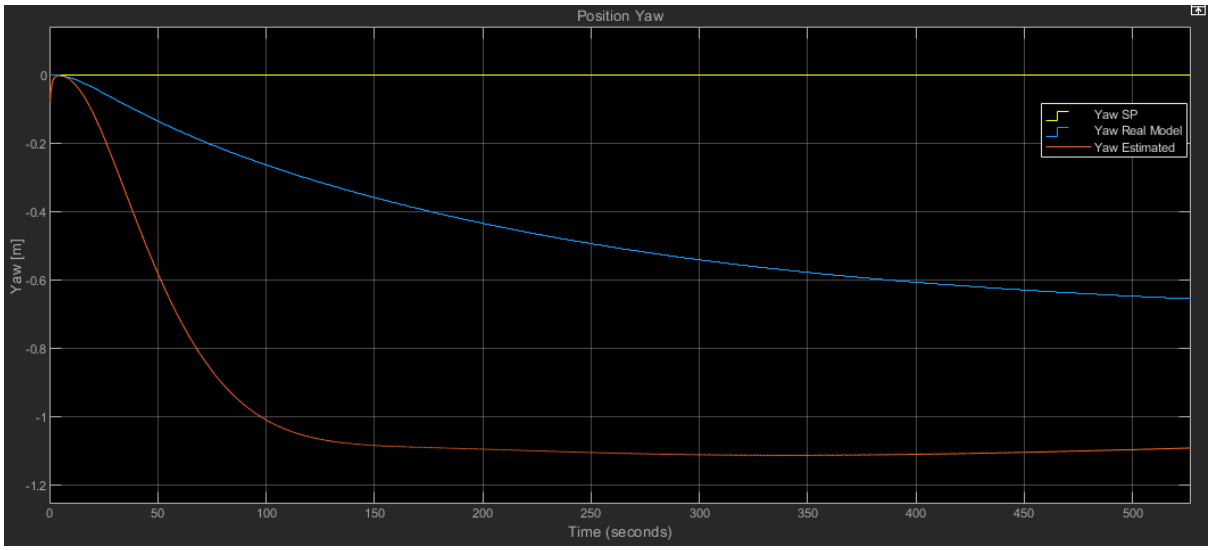


Figure 5-37 Simulation with SP=10 in north direction and SP=10 in east direction. Here are the results for yaw. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

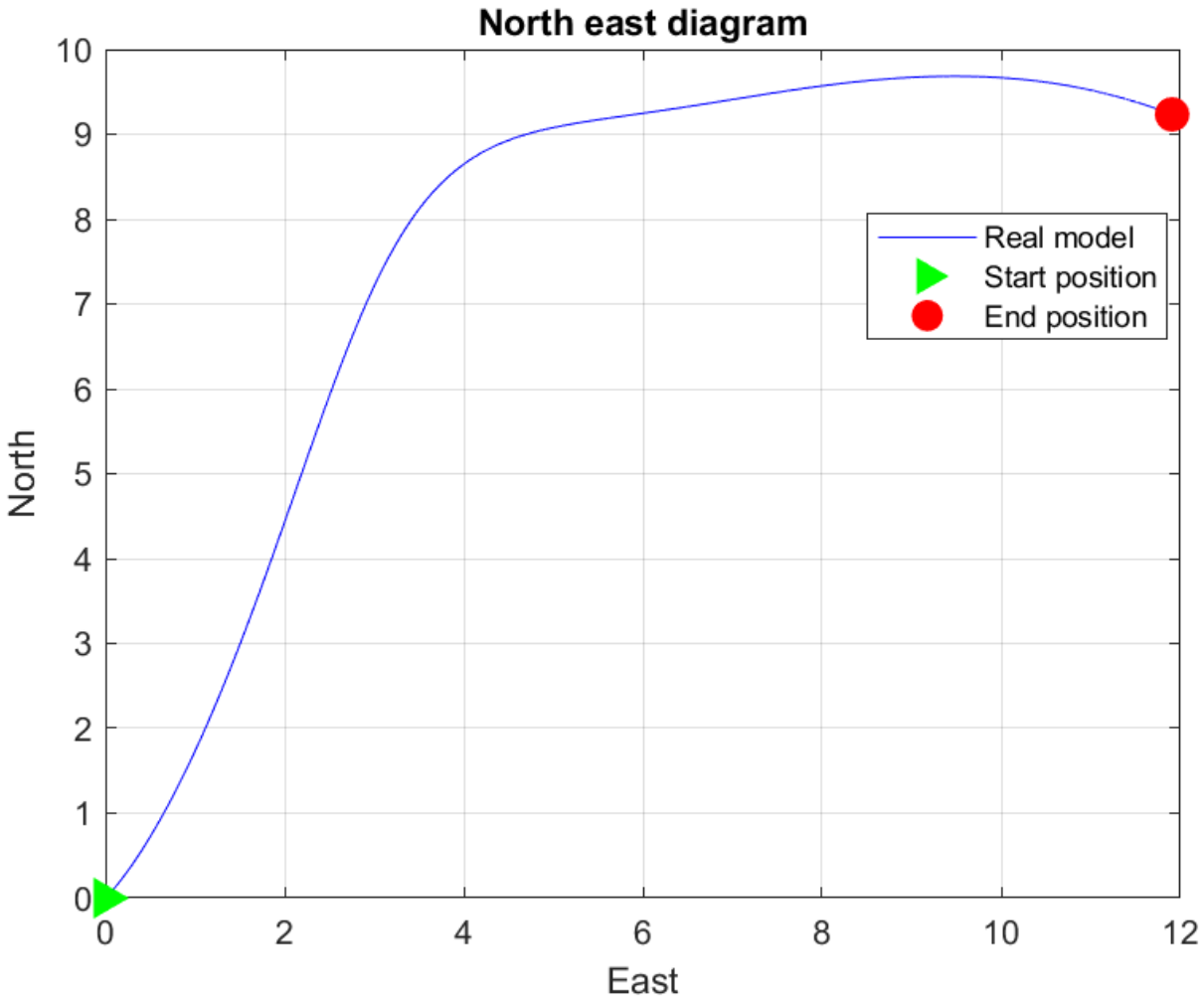


Figure 5-38 NE diagram for simulation with SP=10 in both north and east directions.

Figure 5-39 is included to show how bad the MPC controller performs, and to show the need for improvement. When Gunnerus is going in a negative direction in east, the controller is far off. It was decided to stop trying to adjust weighing from this part and try to develop a better controller.

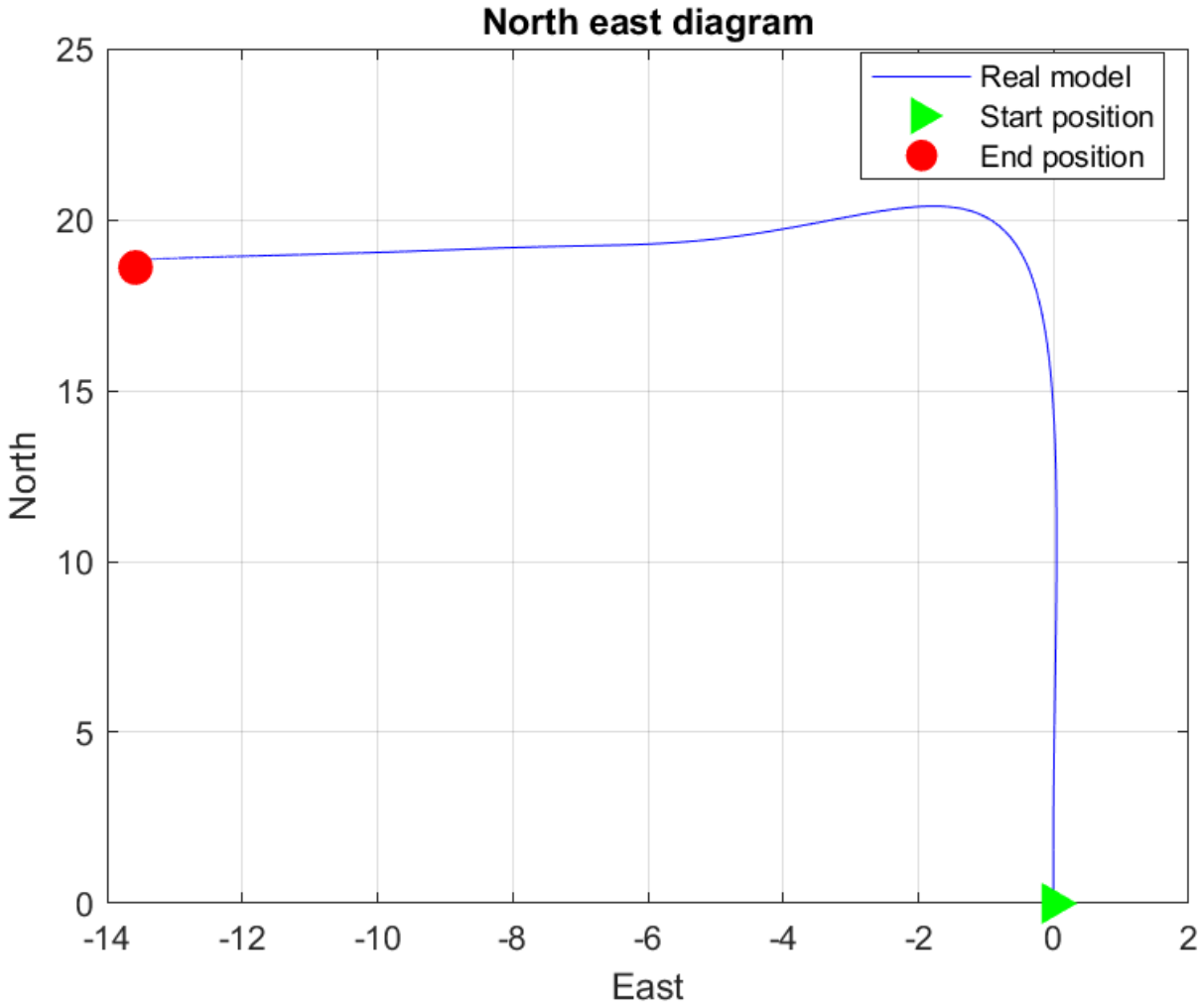


Figure 5-39 Test with SP=-10 in east and SP=10 in north

5.4.2 MPC Controller with integral action

Since the identified model in chapter 3.2 and 5.2 seem to not respond well in yaw, it seems reasonable to put more trust on the weighting matrix Q rather than the weighting matrix P for yaw. This means that the MPC controller is more driven on the error rather than the prediction.

While trying to adjust the integral action for surge direction the weightings in Q and P were adjusted to an unfavorable value in sway and yaw to minimize disturbance from those parts.

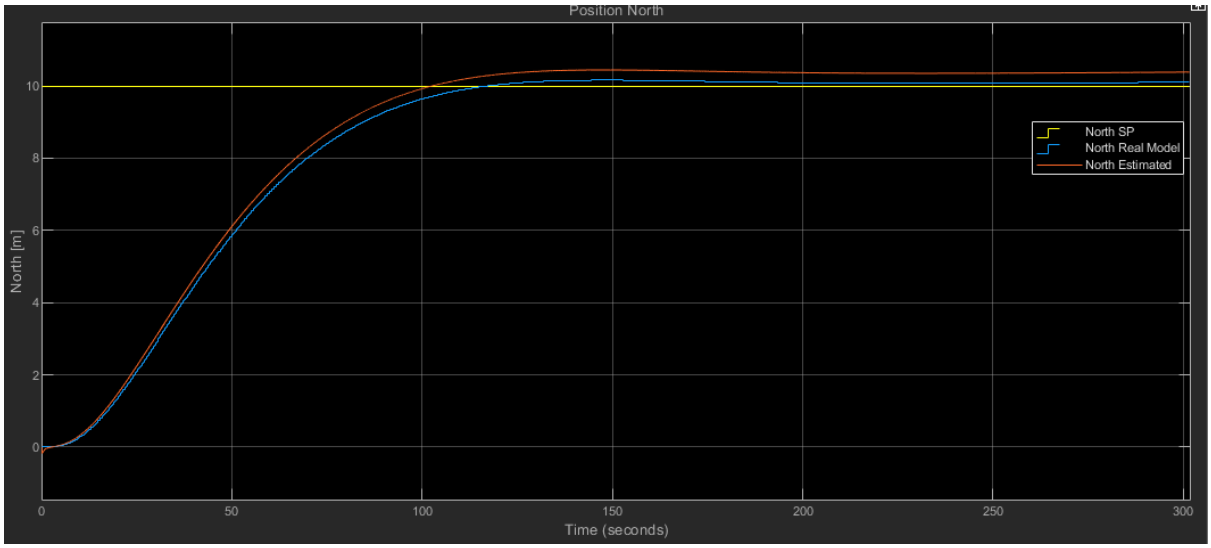


Figure 5-40 Figuring out the integral gain in surge. SP=10 in north direction and SP=0 in east direction. Here the north direction is shown. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

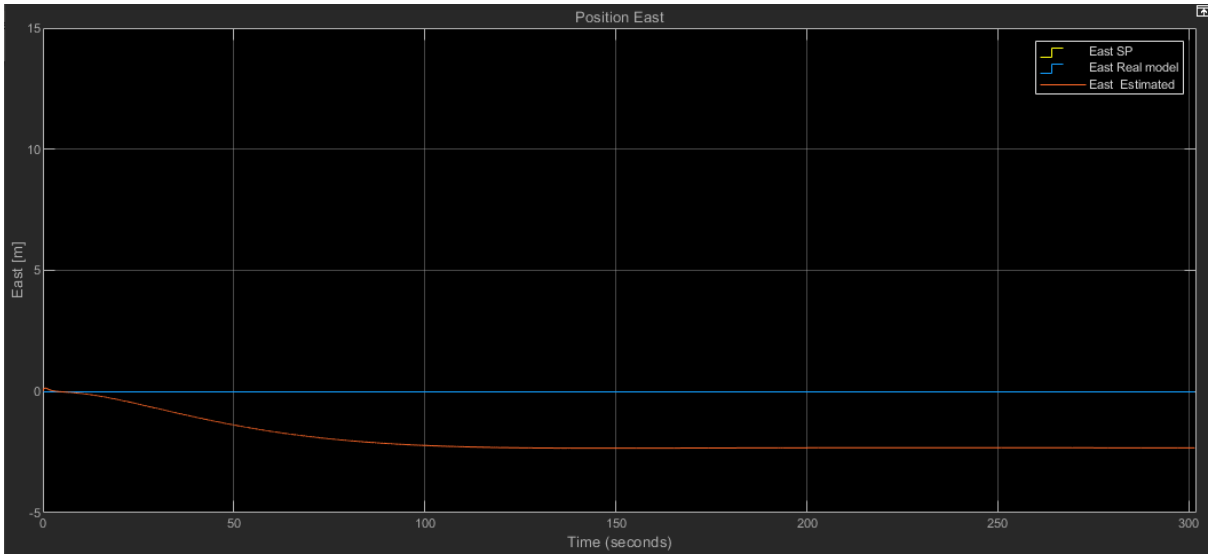


Figure 5-41 Figuring out the integral gain in surge. SP=10 in north direction and SP=0 in east direction. Here the east direction is shown. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

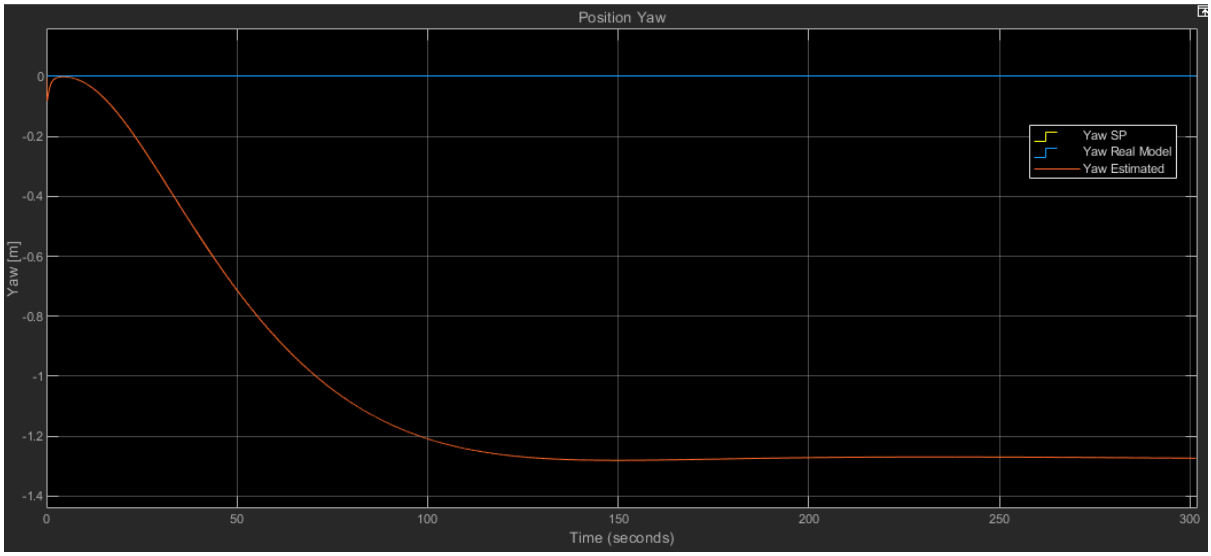


Figure 5-42 Figuring out the integral gain in surge. SP=10 in north direction and SP=0 in east direction. Here the yaw angle is shown. Notice how far off the estimation is. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

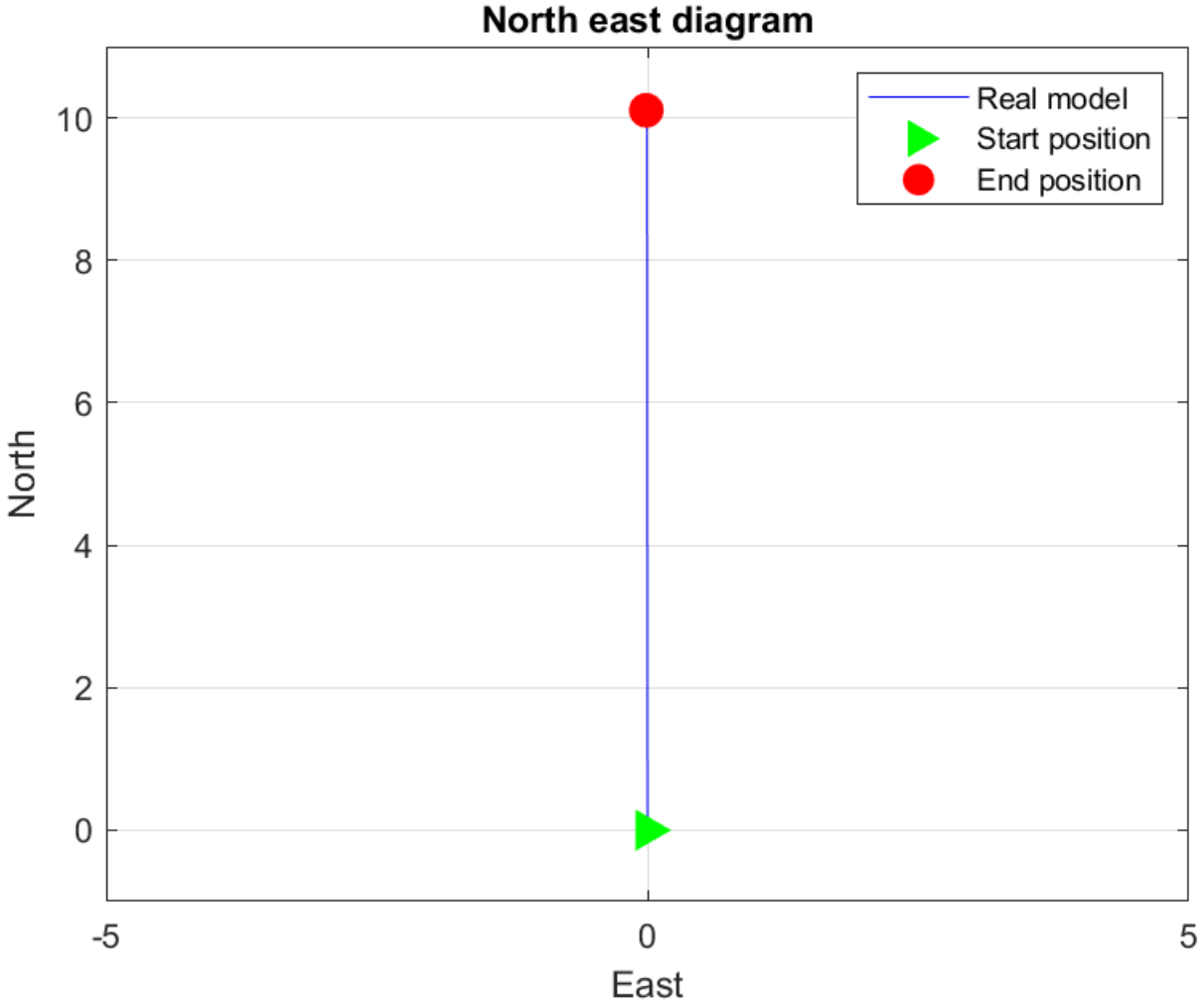


Figure 5-43 NE diagram with SP=10 in north direction and SP=0 in east direction.

While trying to adjust the integral gain in sway and yaw it became clear that the state estimations where not good enough to feed back to the Linear MPC block from Figure 3-18. Therefore, the true measurements were added to the feedback loop together with the velocity estimates in surge, sway, and yaw. This was done in the State Estimator depicted in Figure 3-18 and Figure 5-44 shows the result. Notice that there is a unit delay for each signal to avoid algebraic error due to initial values not being available at startup.

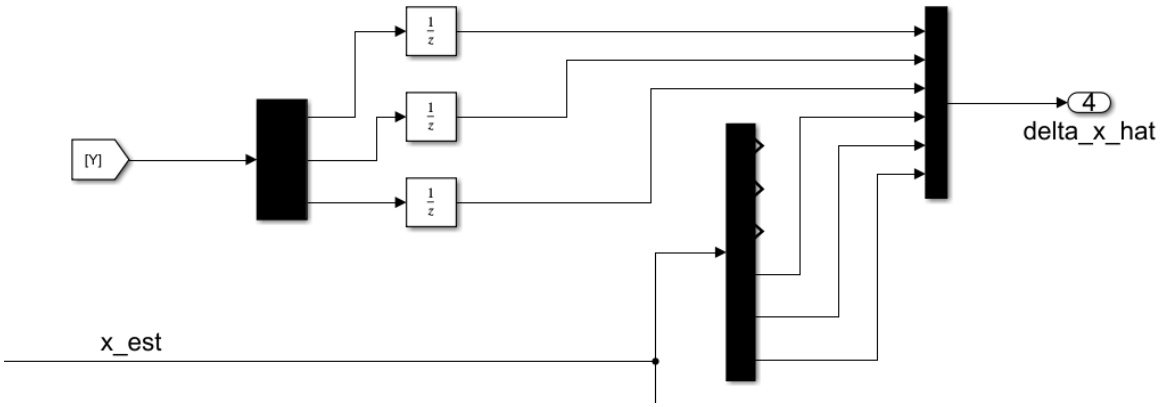


Figure 5-44 The true measurements added as feedback in delta_x_hat as well as the estimates in velocity for surge, sway, and yaw.

The results in Figure 5-45, Figure 5-46, and Figure 5-47 shows the result with SP= 5 in north and east, and SP=0 in yaw. For yaw it looks like the integral action is performing badly so it will be increased. For the north direction it seems like there is a bit of overshooting and the weightings will be adjusted to be more sensitive to error. The error in surge seems partly to stem from wrong integral action as well as wrong weighting setting in the error weight Q. But overall, the MPC controller seemed to perform better when the measurements are inputted directly to the MPC controller.

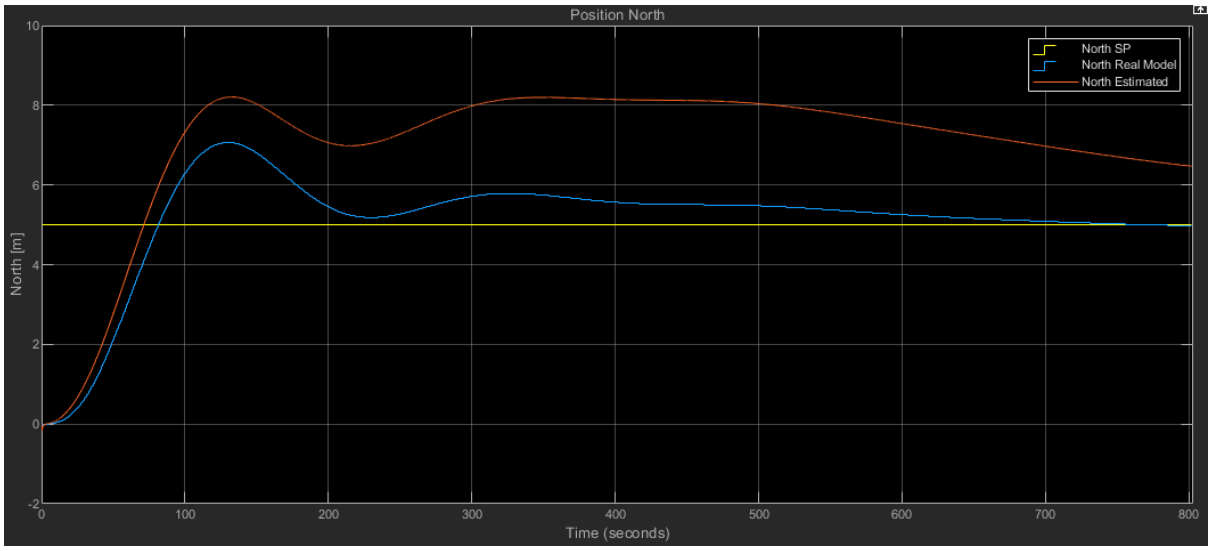


Figure 5-45 Integral action in north direction with a bit of overshoot. SP=5 in north and east. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

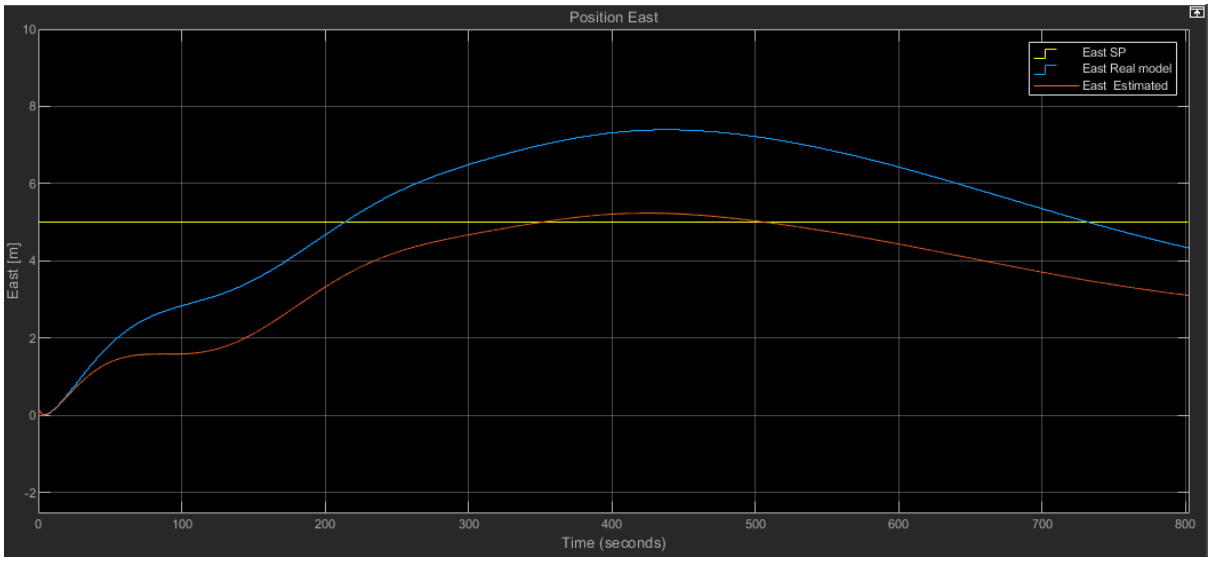


Figure 5-46 Integral action in east direction with a bit of overshoot. SP=5 in north and east. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

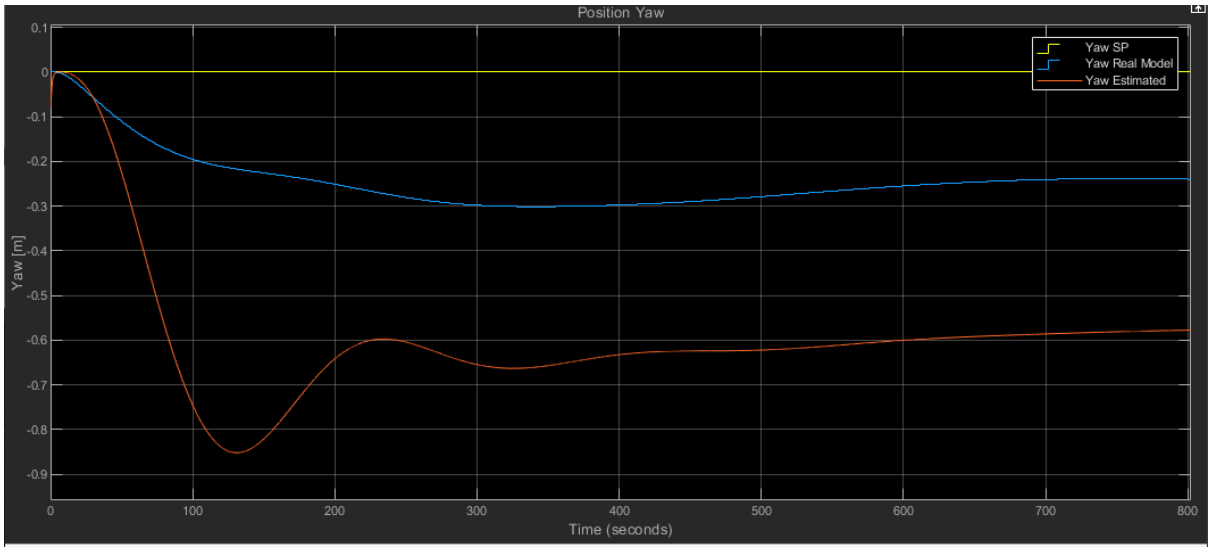


Figure 5-47 Integral action in yaw, notice how far off the yaw is for the real model. SP=5 in north and east. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

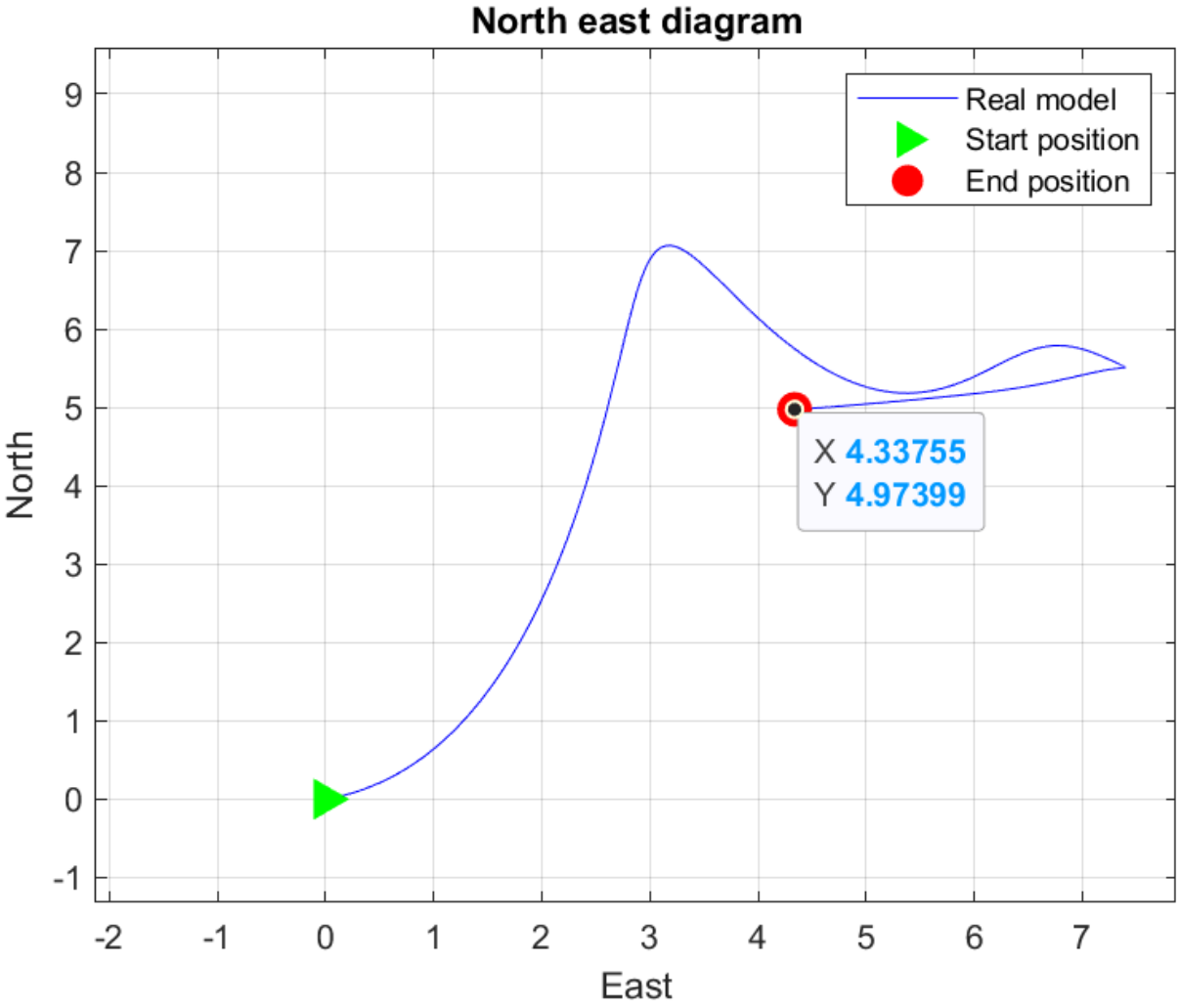


Figure 5-48 NE diagram with a lot of overshoots in north and east direction.

Figure 5-48 shows the overshoot more clearly, here it seems like the marine vessel is moving more than needed to stabilize at the SP. The next experiment will focus on how to adjust the weighting matrices Q and P, so that the weighting matrix Q is more sensitive to error. The new set points are now SP=5 in north and east, and SP=0 in yaw. The weighting settings are shown in Figure 5-49.

```
7
8       N = 20;                          %Prediction horizon length
9       Q = diag([3e4 5e4 1e4]); %Weighting matrix for the error
10      P = diag([2e-6 2e-6 1e-6]);% Weighting matrix for the input
11
```

Figure 5-49 The weighting settings for SP=5 in north and east, and SP=0 in yaw.

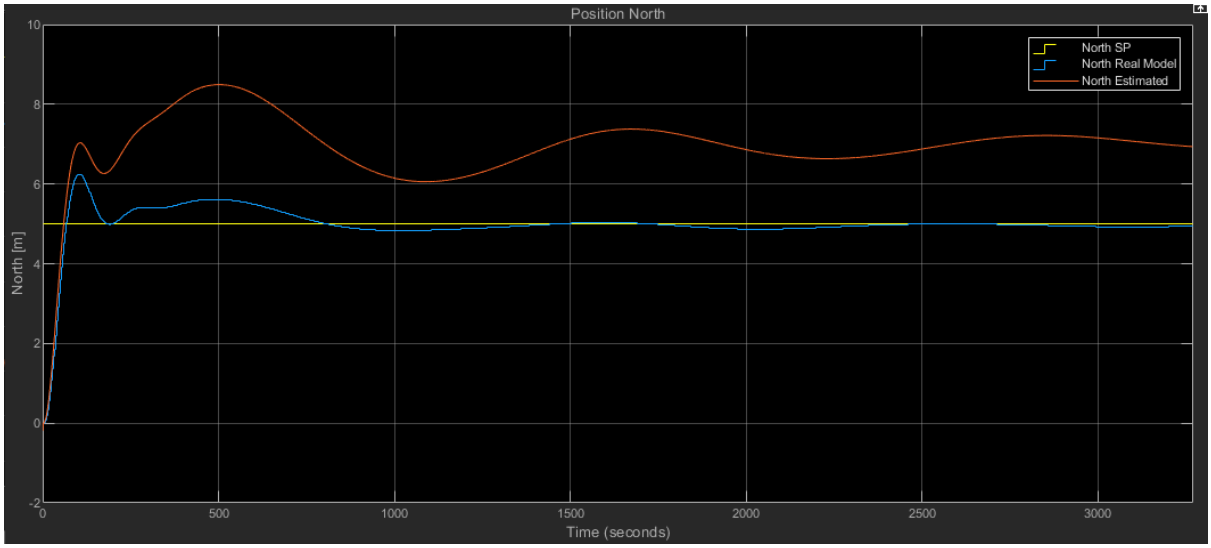


Figure 5-50 Here the position in north is shown and clearly there is sufficient integral action, and the weighting settings makes the controller behave well. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

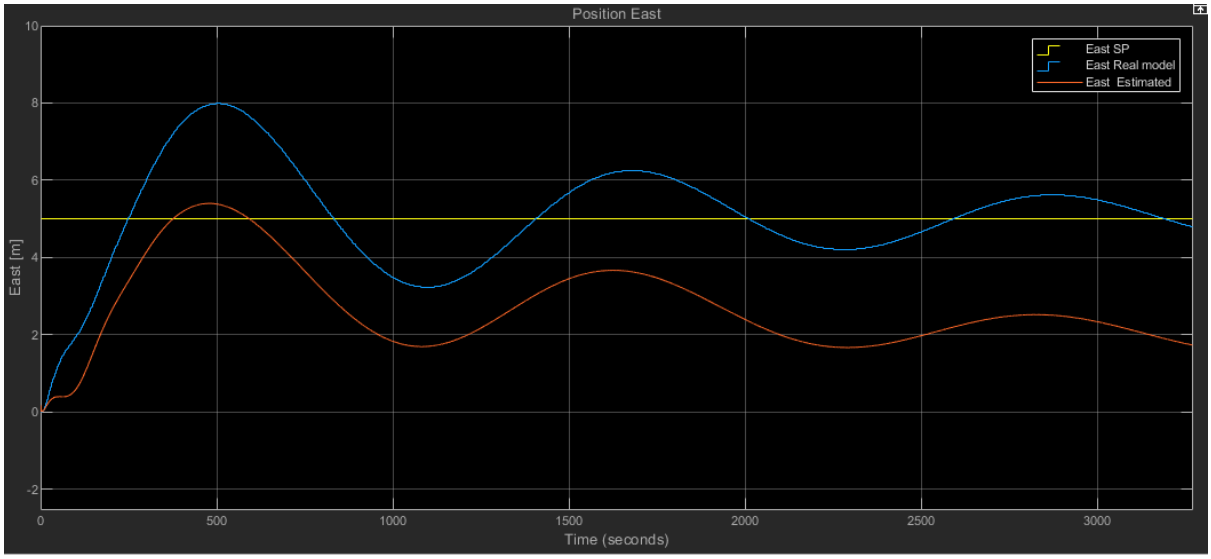


Figure 5-51 Here the position in east is shown and there seems to be sufficient integral action. The weighting settings for P and Q does make the controller perform badly in sway and could be adjusted to be more sensitive in the error matrix Q. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

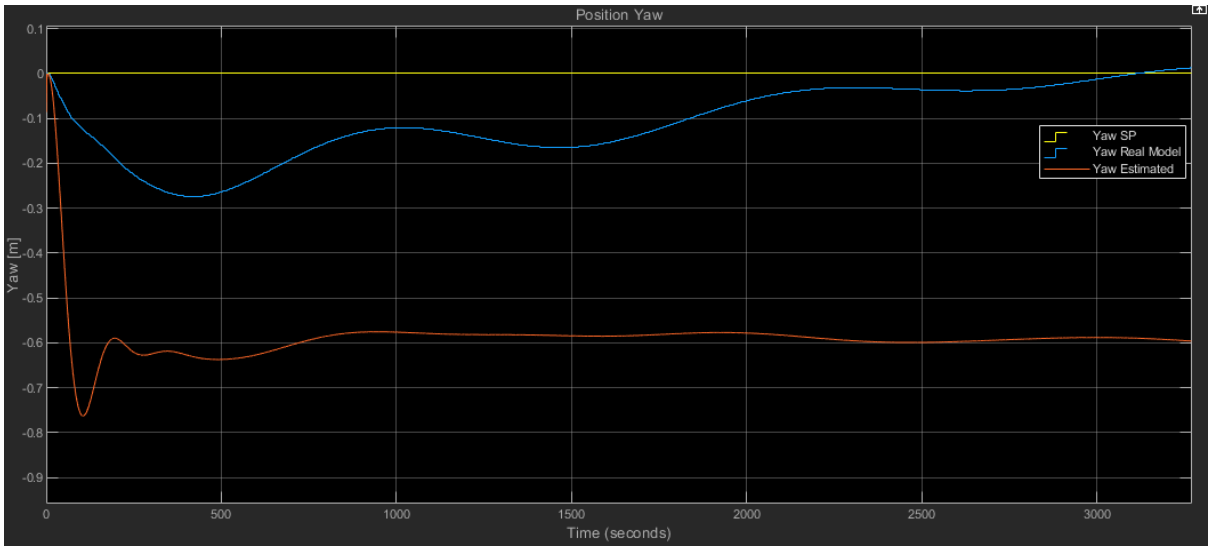


Figure 5-52 Here yaw is shown, and the weightings might be correct, but the integral action uses a long time to adjust. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

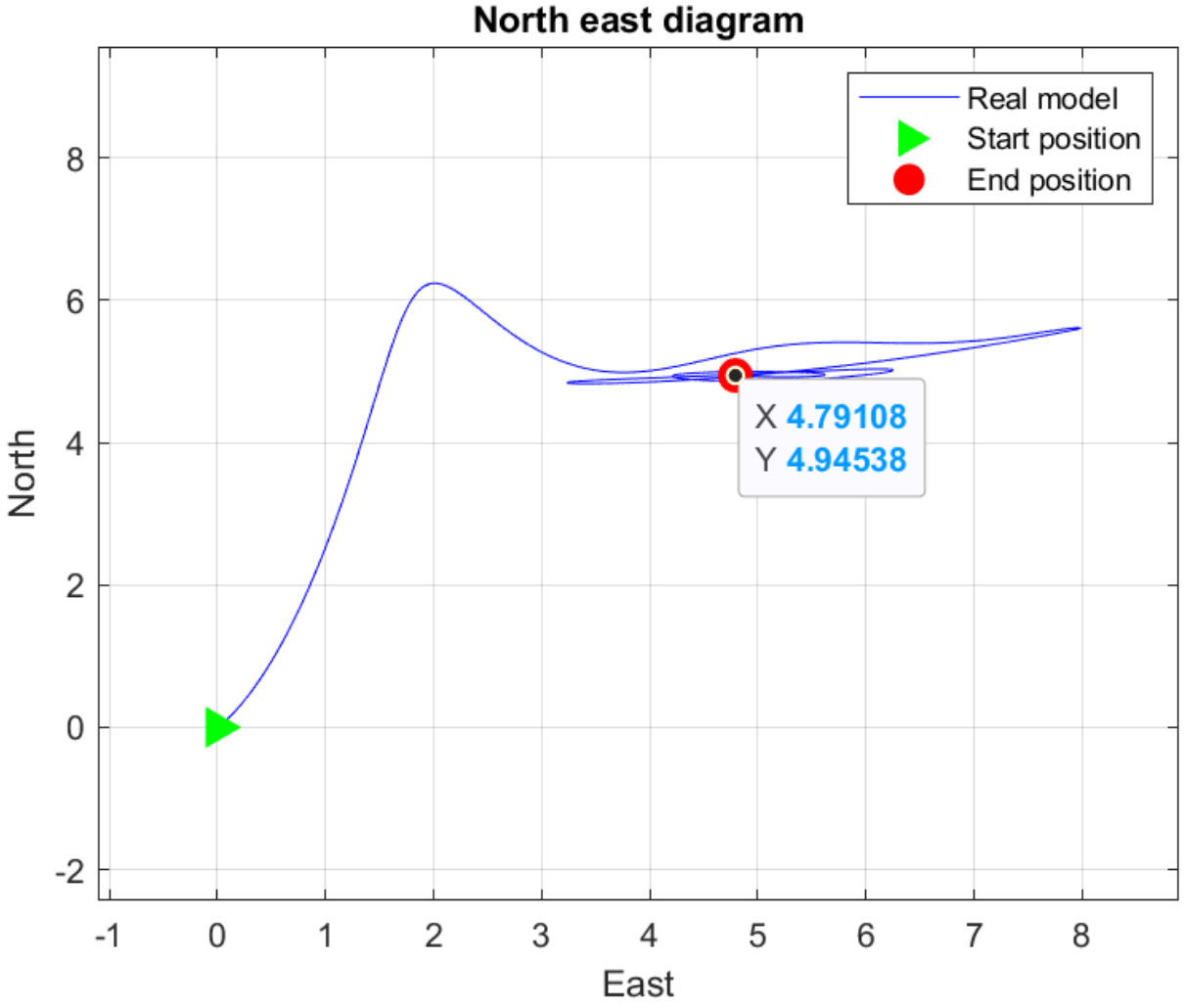


Figure 5-53 In the NE diagram the oscillating effect in east becomes much more apparent. This oscillating effect can be removed by adjusting the weights Q and P.

The previous results were not satisfying, especially the oscillating effect in the east direction. Adjusting the weights so that the controller is more sensitive to errors in east direction, as well as north to decrease the overshooting effect. The gains in integral actions are now 23 in surge, 8 in sway, and 20 in yaw. The weightings are shown in Figure 5-54 and the model now relies more on the error rather than the predicted output.

```
8    N = 20;                          %Prediction horizon length
9    Q = diag([4e4 9e4 8e4]);         %Weighting matrix for the error
10   P = diag([1e-6 1e-6 1e-6]); % Weighting matrix for the input
11
```

Figure 5-54 The weighting settings relying more on the error rather than the model prediction.

For this simulation the SP=5 in north and east, and SP=0 in yaw.

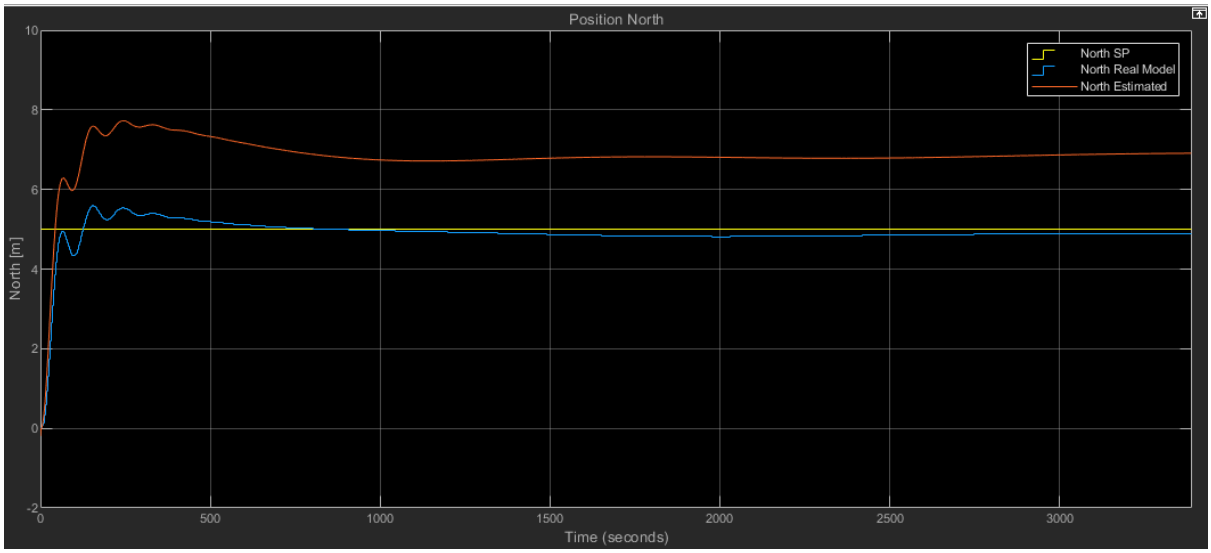


Figure 5-55 Here the position in north is shown and the controller behaves well in both integral action and the weighting settings. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

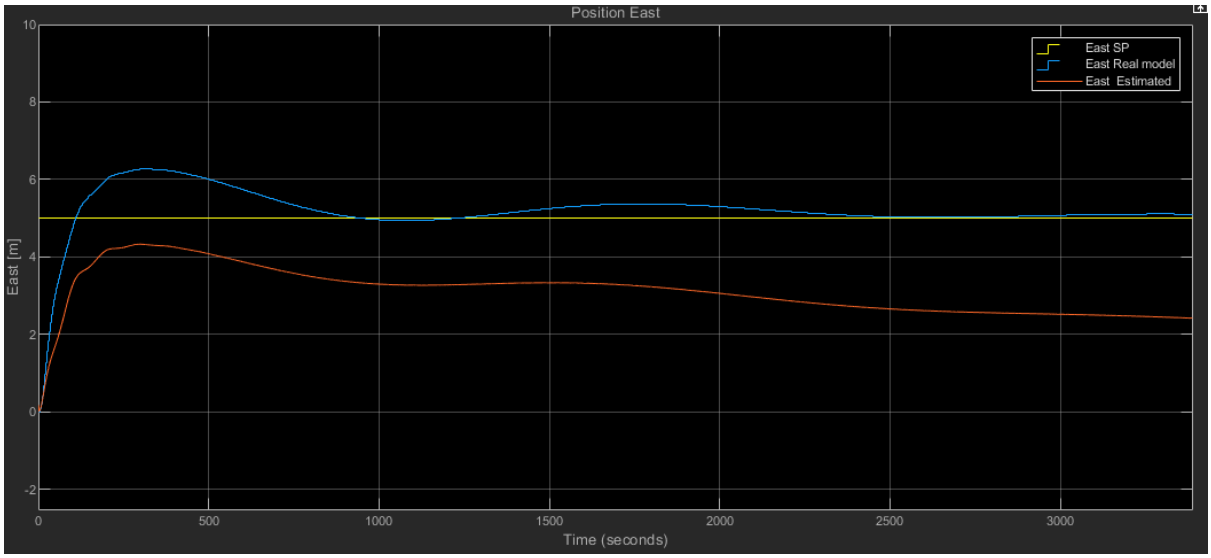


Figure 5-56 Here the position in east is shown and the controller behaves well with integral action, but there is still some overshoot. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

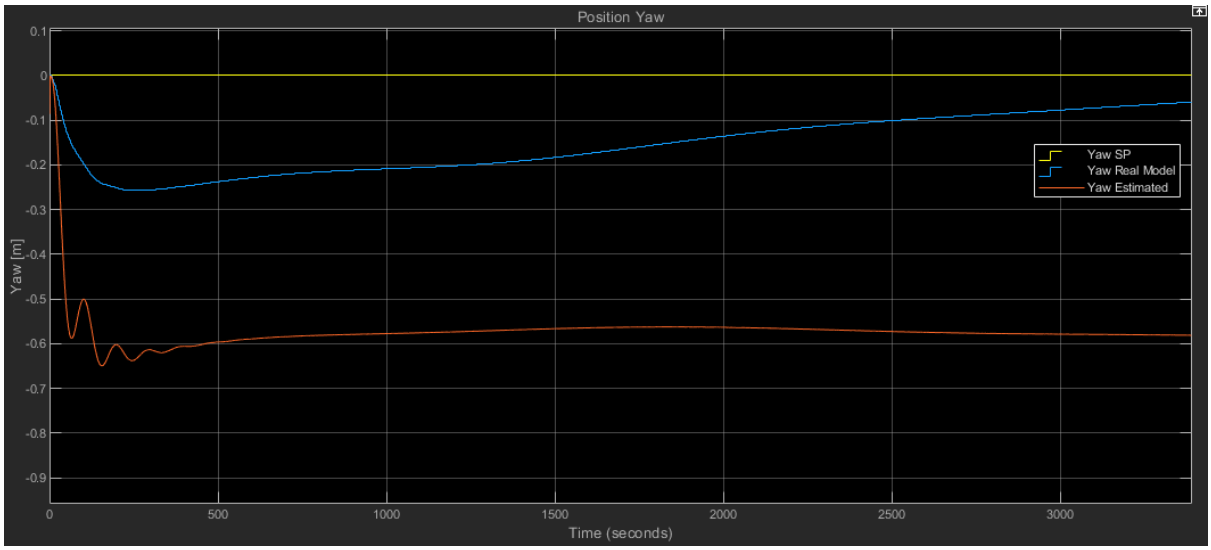


Figure 5-57 Here yaw is shown, and it still takes a long time for yaw to settle at SP. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

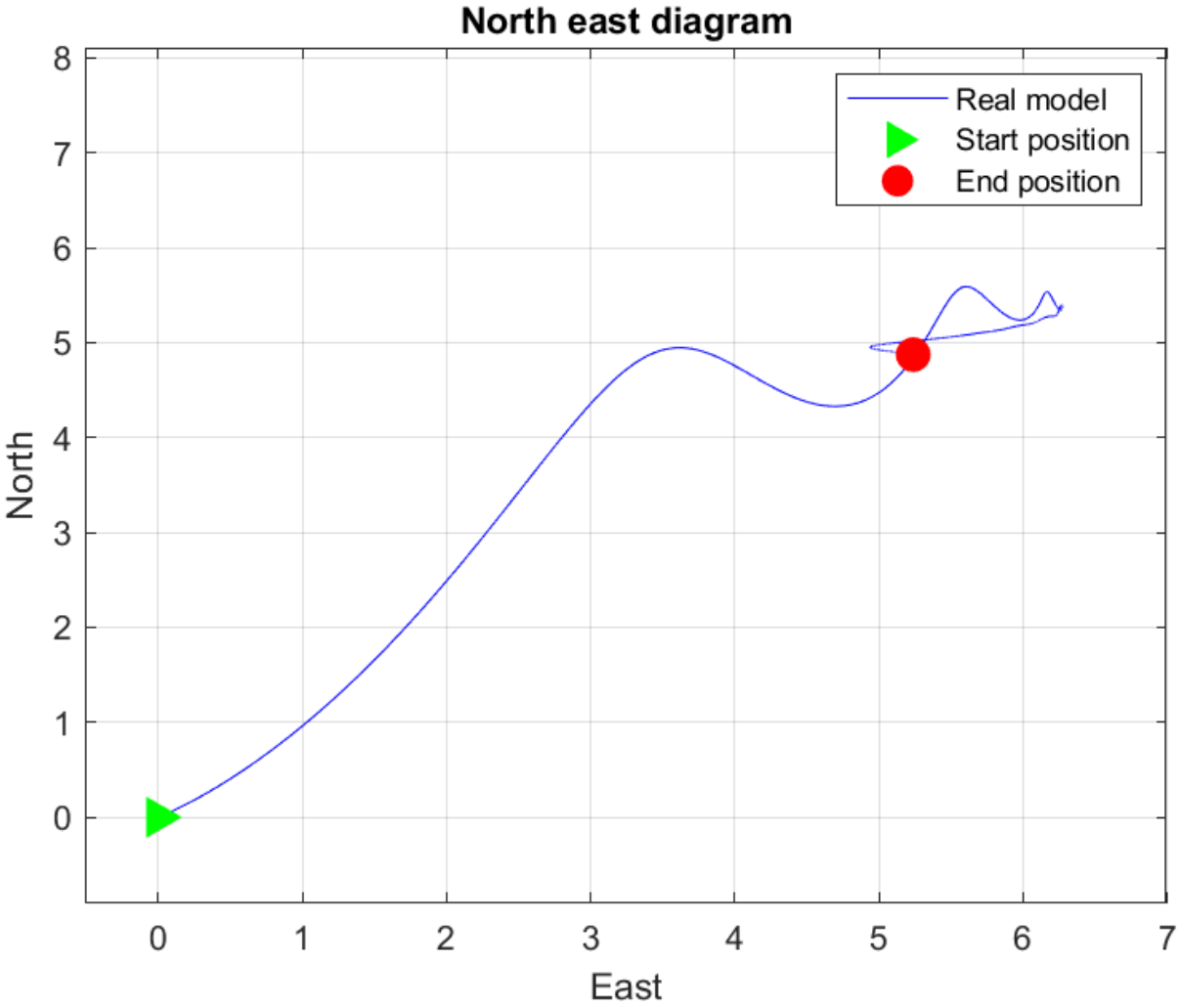


Figure 5-58 Here is the NE diagram for SP=5 in north and east, and SP=0 in yaw. There is very little overshoot, and the overshoot is mainly in east direction.

As the simulation experiments for a MPC controller without integral action shows in Figure 5-38 and Figure 5-39 the results in a negative direction where much worse than in a positive direction in the NE diagram. A new experiment was performed to figure out how the MPC controller with integral action performs in a negative direction. Here the SP=-5 in north and east, and 0 in yaw. The weighting matrices P and Q are shown in Figure 5-59 and the results are shown in Figure 5-60, Figure 5-61, and Figure 5-62.

```
7
8   N = 20;                          %PRediction horizon length
9   Q = diag([4e4 9e4 8e4]);        %Weighting matrix for the error
10  P = diag([1e-6 1e-6 1e-6]); % Weighting matrix for the input
11
```

Figure 5-59 The weighting settings for SP=-5 in north and east, and SP=0 in yaw.

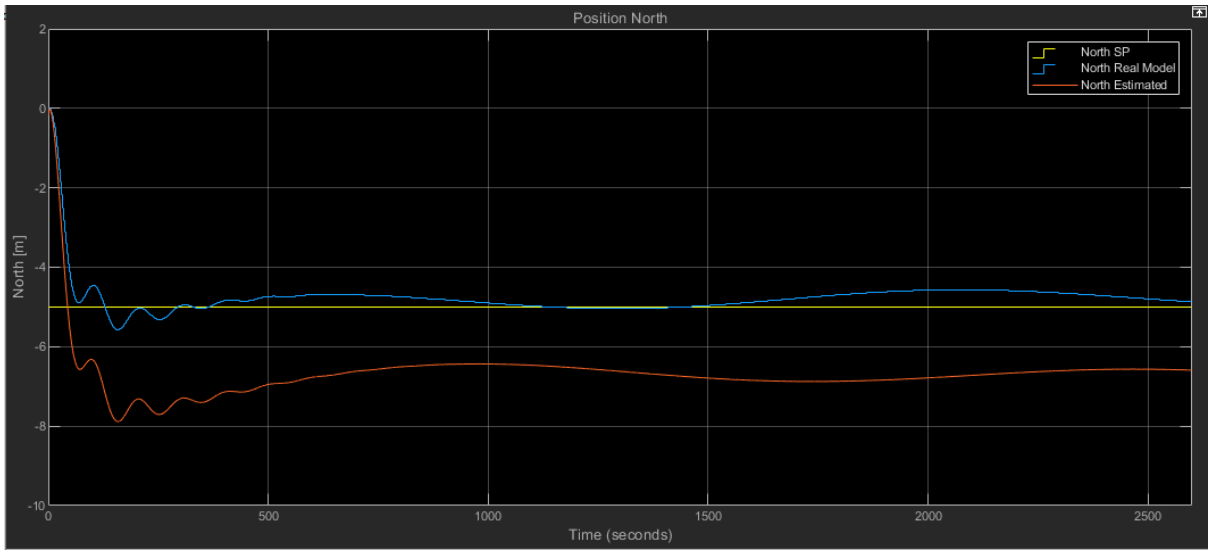


Figure 5-60 Here the position in north is shown and both the weightings and the integral action seems to perform well. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

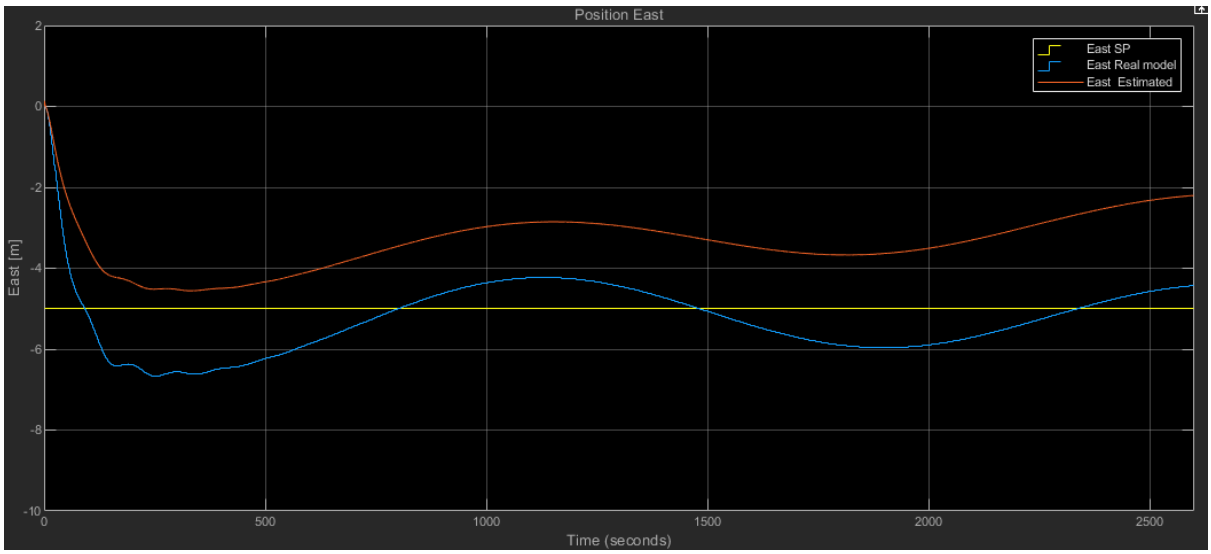


Figure 5-61 Here the position in east is shown and the weighting settings seems like they can be adjusted a bit in favor of the error matrix Q to compensate for the overshoot. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

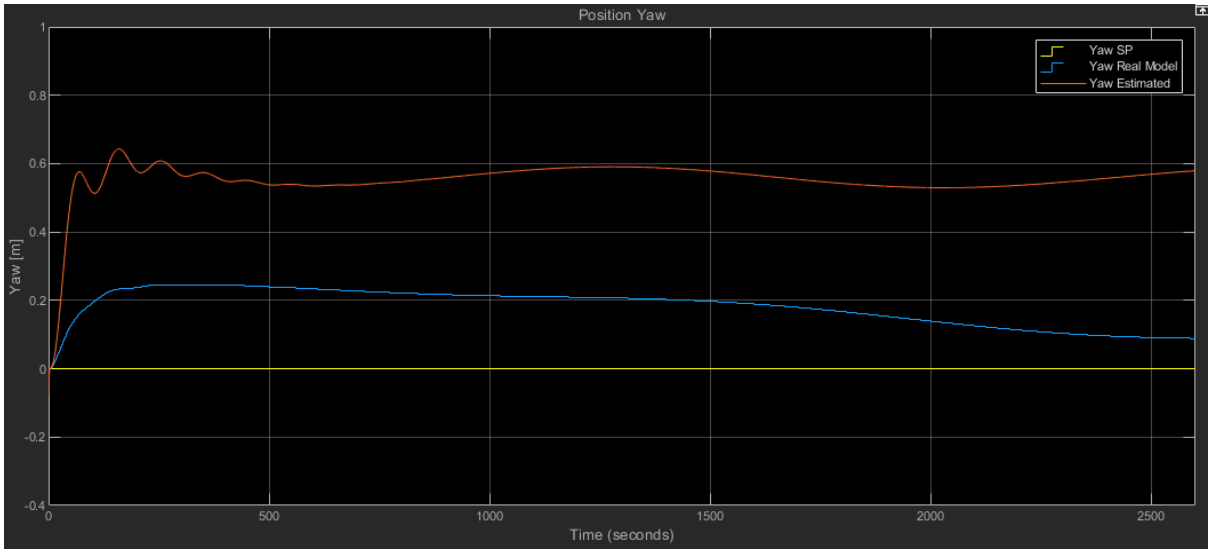


Figure 5-62 Here the yaw is shown, and it seems like the integral action is working but it takes over 2500s to adjust. The yellow line is SP, the blue line is the real model, and the orange line is the estimated value.

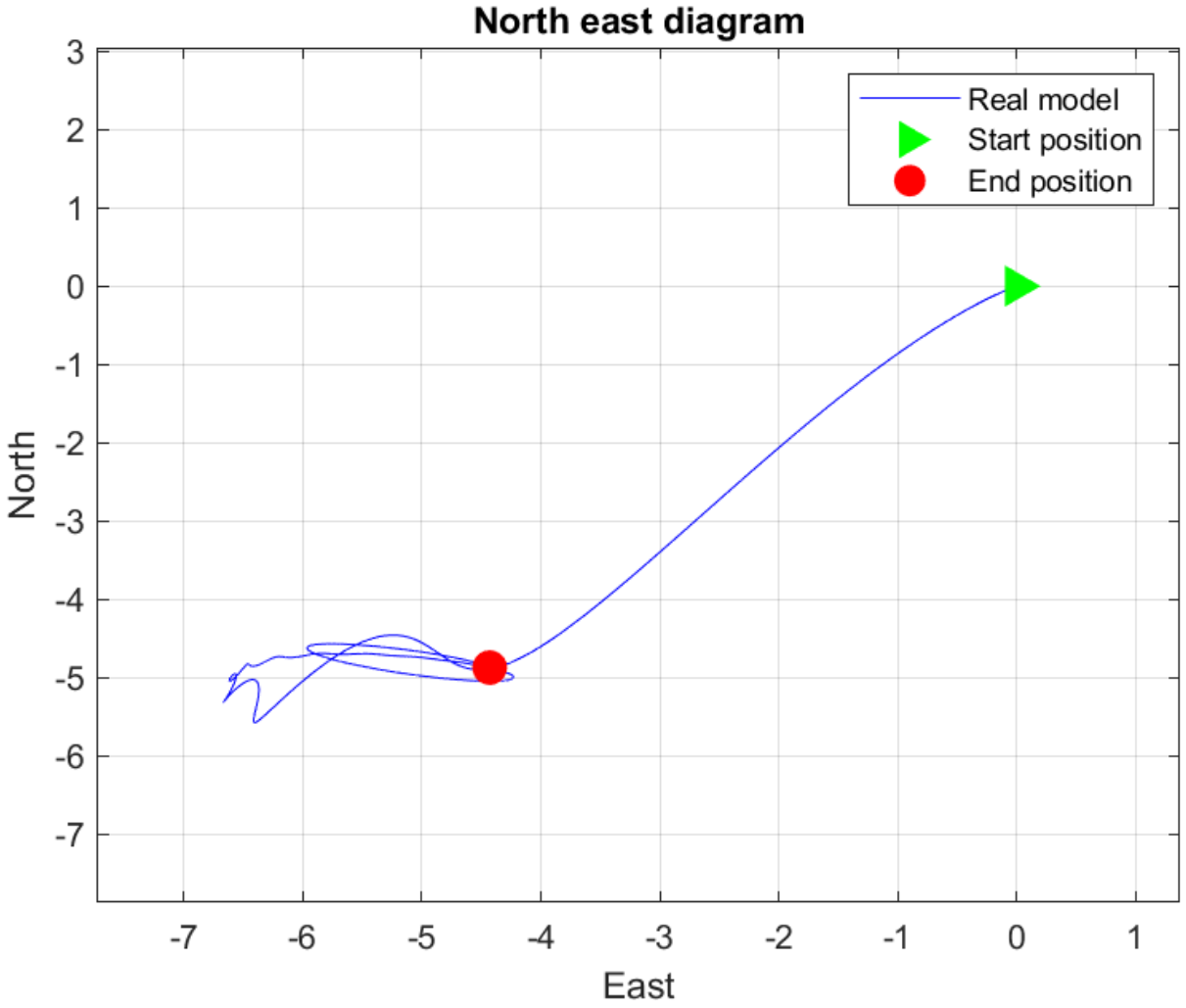


Figure 5-63 In the NE diagram the overshoot in east direction is much clearer.

6 Discussion

One of the earliest problems that occurred was that there was a mismatch between the required inputs in the model according to the documentation and the provided model in the OSP package. According to the documentation the vessel model required 6 inputs in thrust, 6 inputs in wave force etc. But when the simulation started it seemed like the model needed 7 input thrust, wave force etc. This was a strange problem, but it seems like the model uses index 1-6 or 1-4, but when imported as an FMU it also requires the index 0. The index 0 doesn't seem to do anything of importance in the simulation, and it doesn't behave strangely when inputting different signals to it. The solution was to add an extra signal as index 0, and this signal was set to the constant 1.

Since this was a provided model from the OSP package which seemed to accept every input force and create an output that doesn't always make sense, it was decided to research the engine power output. The datasheet for the RV Gunnerus [19] states that each main propeller produce 500kW each, a total of 1000kW. The datasheet also specifies a 100% MCR at 12.6kn which results in about 6.842m/s. The thrust that the propeller produces will depend on the efficiency of the propeller and since there is no data about the propeller efficiency it is hard to calculate the thrust the propellers will produce. The estimate in equation (3.1) doesn't seem to fit with the Gunnerus-DP package, as the velocity is closer to 2m/s at that force. It was decided from this point to not do further investigation on this part and continue with forces up to 265750N in all inputs.

During the system identification there were some difficulties finding an accurate model. The Random source blocks in Simulink are discrete time as default. This means that the user can set a time interval which the block should update its random value. This also affected the vessel model which then behaved as a discrete model. This means that if the sample time for new values was set to 10 seconds, the vessel model only updated the outputs every 10 seconds even though the model solver was selected to be Runge-Krutta with a timestep of 0.1s. The solution for this was to make the Random Source block into a continuous time signal, but then the square pulses were locked to 0.1 timestep from the model settings.

There is also a problem with the identification of the SSM for the vessel model since DS-R only finds a discrete linearized model. Therefore, the system identification must be limited either by time or force sent to the propeller to keep the vessel model close to Origo. Otherwise, the system identification process is unsuccessful, and the identified model is a horrible match. Another issue trying to find a linearized model is that when a successful SSM model is found, it doesn't behave very well when the model moves further away from origo in north and east. This is likely due to 0 north and 0 east is the operation point for the linearized model, and the vessel model includes nonlinear parts.

For the MPC controller it seems like the controller behaves well navigating in the north direction but is slower in the east direction. This can be due to the model not being accurate in east, or that the weighting matrices Q and P needs to be adjusted more.

As Figure 5-26, Figure 5-27, Figure 5-28, and Figure 5-29 shows there are some deviations in north direction, and the controller is tuned to be more smooth so that it doesn't overshoot before stabilizing close to the SP. These deviations are likely due to the MPC controller being based on a linear model, and the deviations might be removed with integral actions. The same

goes for Figure 5-31, Figure 5-32, and Figure 5-33 showing the results with a SP= 10 in east direction.

When reaching for a SP=10 in both east and north the model has less deviations. This is shown in Figure 5-38. The reasons for this can be many, but it is likely due to the estimated model being a better fit for change in both north and east direction due to system identification process. When trying a SP=-10 in east direction, and SP=10 in north direction the model seems to have bigger deviations in SP as Figure 5-39 shows.

To make a better model it might be needed to convert the vessel model from NED coordinates to BODY coordinates. The reason for this is because the MPC controller knows exactly that a thrust change in surge gives a change in surge position. The same goes for thrust in sway and sway position in BODY diagram.

Another thing worth mentioning is that the control values from the MPC controller had to be multiplied by -1. This was discovered due to the model acting in opposite directions of the selected SP.

When developing an MPC controller with integral action several adjustments were made to make the controller perform sufficiently. Since the state estimator wasn't accurate in estimating yaw it was decided to use the measurements in north, east and yaw directly together with the velocity estimates in surge, sway, and yaw. This made the controller perform better and now more trust could be put in the error weighing matrix Q to adjust for overshoot for example. The integral action performs very well in removing the deviation from SP, and together with weighting settings in favor of the error matrix Q the MPC controller with integral action performs well.

At the end of the project phase, it was discovered an error in the OSP documentation regarding yaw, and this error is just a unit error. This error got into the report for some of the plots in yaw, and the correct unit should be [rad]. This makes sense since an angle is not a linear measurement.

When looking back at the found system and MPC controller with integral action one drawback is the time the controller uses to stabilize and remove the deviation, especially in yaw with integral action. One solution to this could be to further explore weighting settings for the controller and see if there exist better weight settings for the error matrix Q.

6.1 Further work

Since the OSP is designed for co-simulation[10] and the reference models are used as a starting point to use the OSP package, it is a good idea to develop a new model to use for further work on this topic. The ship can still be the Gunnerus, but then the measurements for system identification should be taken on the real model. Otherwise, there can be taken data from other models, or even the Balchen model and build this model into an FMU for use with the OSP simulator.

There could also be made a vessel model based on the Balchen model that runs in parallel with Gunnerus to check for similarities. This can be useful to find similarities which can be used to develop a DP system that is flexible and can be used on other marine vessels.

A further improvement that can be made directly to this work is to tune the MPC controller with integral action so that it responds faster. This is useful for the operators of the marine vessel to minimize the time to reach SP. This would likely require adjusting the weightings in Q and P, probably in favor of Q so it can compensate more for deviation from SP.

7 Conclusion

The primary objective of this project was to do literature research of DP systems, use an existing dynamic model to perform system identification on and implement a DP system for the selected model. Part of the work done to reach this goal was to implement testing of the selected dynamic model, design a state estimator and experiment with different controller setup. MATLAB and Simulink were used to perform simulation experiments.

While the theory chapter gave insight into different topics, the practical work was mainly from the methods and result chapter. Here the work was divided into multiple parts consisting of; open loop testing to see how the model behaved, system identification to find a model, state estimator to find the unknown states, and lastly the design of a controller to keep the marine vessel at certain position.

There were two alternatives in regards of the model selection, but the final decision was made towards the Gunnerus model from the Gunnerus-DP reference model in the OSP package. The reason for this was to explore the OSP package and the Balchen model has been used in previous projects. This way the project was more in line with the background for this task.

Based on the simulation experiments the most successful controller for the DP system was found to be a MPC controller with integral action. The MPC controller that was found gives very little deviations as shown in Figure 5-58 for example, and this can be acceptable for many marine applications. As with the problems with the identified model showed, most of them could be compensated for as presented in the discussion chapter.

There is still room for improvements as mentioned in the discussion chapter. Directly related to this task is the tuning of the weighting matrices Q and P for the MPC controller. This can decrease the time the yaw uses to reach SP and make the controller more suitable for situations where the angle of the marine vessel is important.

The final DP system also aligns well with the requirements and design chapter where the requirements where explored, and the controller behavior was analyzed on a top layer. As depicted in the result chapter the controller delivers adequate performance when changing SP and this is crucial for a DP system.

References

[1] “The story behind dynamic positioning.” Accessed: Jan. 21, 2024. [Online]. Available: https://www.kongsberg.com/kmagazine/2014/3/story-behind-dynamic-positioning/

[2] J. G. Balchen, N. A. Jenssen, E. Mathisen, and S. Sælid, “A dynamic positioning system based on Kalman filtering and optimal control,” p. 29, 1980.

[3] N. M. Bargouth, “Dynamic positioning, system identification and control of marine vessels,” Master Thesis, University of South-Eastern Norway, Porsgrunn, 2022. Accessed: Jan. 24, 2024. [Online]. Available: https://openarchive.usn.no/usn-xmlui/bitstream/handle/11250/3000363/no.usn%3Awiseflow%3A6583421%3A50226129.pdf?sequence=1&isAllowed=y

[4] “Maritime Reference Models,” Open Simulation Platform. Accessed: Feb. 11, 2024. [Online]. Available: https://open-simulation-platform.github.io/demo-cases

[5] “Introduction to Dynamic Positioning (DP) Systems.” Dec. 2019. Accessed: Sep. 02, 2024. [Online]. Available: https://www.dco.uscg.mil/Portals/9/OCSNCOE/References/Custom-Ref-Books/Intro-to-DP-Systems-Dec2019.pdf?ver=d1Z9tUwX9p__Mi05A_NkwA%3D%3D

[6] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*, Second Edition. John Wiley & Sons Inc, 2021.

[7] T. I. Fossen, “Lecture Notes TTK 4190 Guidance, Navigation and Control of Vehicles.” [Online]. Available: https://www.dropbox.com/scl/fi/1hdy5puq6p0nzkd4bivtg/Ch10.pdf?rlkey=ro2t6otv1qfn6qg20oi0nxapr&e=1&dl=0

[8] “Gunnerus-DP,” Open Simulation Platform. Accessed: Feb. 10, 2024. [Online]. Available: https://open-simulation-platform.github.io/cosim-demo-app/Gunnerus-DP

[9] “About us - Gunnerus Research Vessel - NTNU.” Accessed: Feb. 11, 2024. [Online]. Available: https://www.ntnu.edu/gunnerus/about-us

[10] “Open Simulation Platform,” Open Simulation Platform. Accessed: Feb. 10, 2024. [Online]. Available: https://opensimulationplatform.com/

[11] “MathWorks - Makers of MATLAB and Simulink.” Accessed: Apr. 07, 2024. [Online]. Available: https://se.mathworks.com/

[12] “MATLAB - YouTube.” Accessed: Apr. 07, 2024. [Online]. Available: https://www.youtube.com/@MATLAB

[13] F. Haugen, “State estimation with Kalman Filter(chapter 29, lecture notes).”

[14] R. Sharma, *Lecture notes for the course IIA 4117: Model Predictive Control*. 2019. Accessed: Feb. 18, 2024. [Online]. Available: https://web01.usn.no/~roshans/mpc/downloads/lecture_notes_MPC.pdf

[15] “What is Model Predictive Control? - MATLAB & Simulink - MathWorks Nordic.” Accessed: Feb. 18, 2024. [Online]. Available: https://se.mathworks.com/help/mpc/gs/what-is-mpc.html

[16] “Model Predictive Control.” Accessed: May 05, 2024. [Online]. Available: https://web01.usn.no/~roshans/mpc/

[17] “What is PID Control?” Accessed: Feb. 18, 2024. [Online]. Available: https://se.mathworks.com/discovery/pid-control.html

[18] D. Di Ruscio, *SUBSPACE SYSTEM IDENTIFICATION Theory and applications*. 2022.

[19] “RV GUNNERUS - LNVZ - Datasheet.” NTNU. Accessed: Oct. 03, 2024. [Online]. Available: https://www.ntnu.edu/documents/1262202806/0/Specifications+RV+GUNNERUS.pdf/6a6540e0-00ae-b7a2-a51d-bf365302bf61?t=1584611463564

[20] Man Energy Solutions, *Basic principles of ship propulsion*. Accessed: Apr. 18, 2024. [Online]. Available: https://www.man-es.com/docs/default-source/document-sync-archive/basic-principles-of-ship-propulsion-eng.pdf?sfvrsn=48fc05b5_7

[21] “D-SR.” Accessed: Mar. 09, 2024. [Online]. Available: https://davidr.no/iia2217/d-sr/d-sr_e.html

[22] D. Di Ruscio, *OPTIMAL MODEL BASED CONTROL: System Analysis and Design*. 2022. [Online]. Available: http://davidr.no/iiav3017/syllabus/main_pc_e.pdf

Appendices

Appendix A – Project Description

Appendix B – GitHub Repository

Appendix C – Plotting function for open loop testing

Appendix D – Open loop testing

Appendix E - Open loop testing Vessel Model

Appendix F – System identification.

Appendix G – System Identification control input function.

Appendix H – System Identification input and output matrixes.

Appendix I – System Identification D-SR.

Appendix J – Kalman Initializing.

Appendix K – Plotting real model and estimated model.

Appendix L – Real model and the state estimator.

Appendix M - QP formulation script.

Appendix N – MPC controller

Appendix O - MPC controller with integral action

Appendix A – Project Description

University of South-Eastern Norway

Faculty of Technology, Natural Sciences and Maritime Sciences, Campus Porsgrunn

FMH606 Master's Thesis

Title: Dynamic Positioning, system identification and control of marine vessels - using the OSP simulator

USN supervisor: David Di Ruscio

External partner: None

Task background:
One of the first mathematical models used for Dynamic Positioning (DP) of ships was the Balchen et al 1980 model. This model was used for building a DP system based on Kalman filtering and optimal control. It is of interest to reconstruct this DP system and possibly make a modified version based on system identification. Hence an experiment in surge, sway and yaw (the three moving directions) in order to obtain data for system identification. From the identified model it is of interest to make a modified DP system. The OSP simulator may be used instead of a real ship, i.e. for experiment design and to collect data.

Task description:

1. Perform a literature research about DP systems of ships.
2. Use an existing dynamic model of a marine vessel and perform experiment design for system identification in order to create models of the vessel.
3. Implement a DP system for the vessel.
4. Perform simulation experiments by using MATLAB or similar.

Student category: IIA students

Is the task suitable for online students (not present at the campus)? Yes

Practical arrangements: None

Supervision:
As a general rule, the student is entitled to 15-20 hours of supervision. This includes necessary time for the supervisor to prepare for supervision meetings (reading material to be discussed, etc).

Signatures:

Supervisor (date and signature): 29.01.2024 David Di Ruscio

Student (write clearly in all capitalized letters): Jan-Robin Brustad

Student (date and signature): 29.01.2024 Jan-Robin Brustad

Appendix B – GitHub Repository

The GitHub repository can be found at:

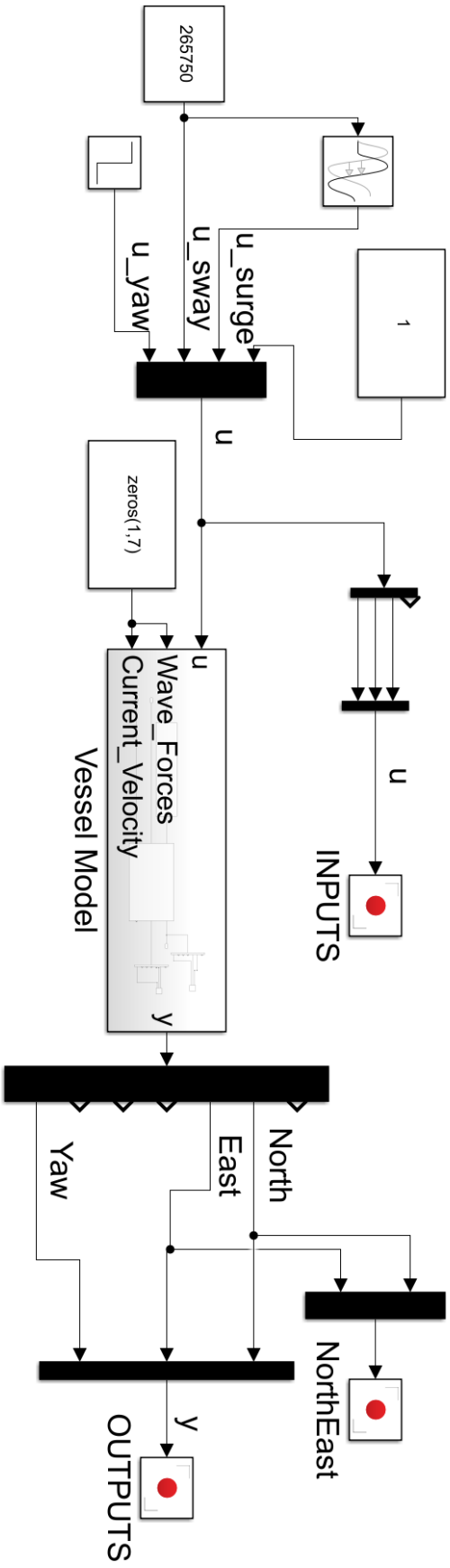
https://github.com/AutomationWithJan/Dynamic-Positioning-system-identification-and-control-of-marine-vessels---using-the-OSP-simulator

Here the different MATLAB scripts and Simulink files from this project will be available.

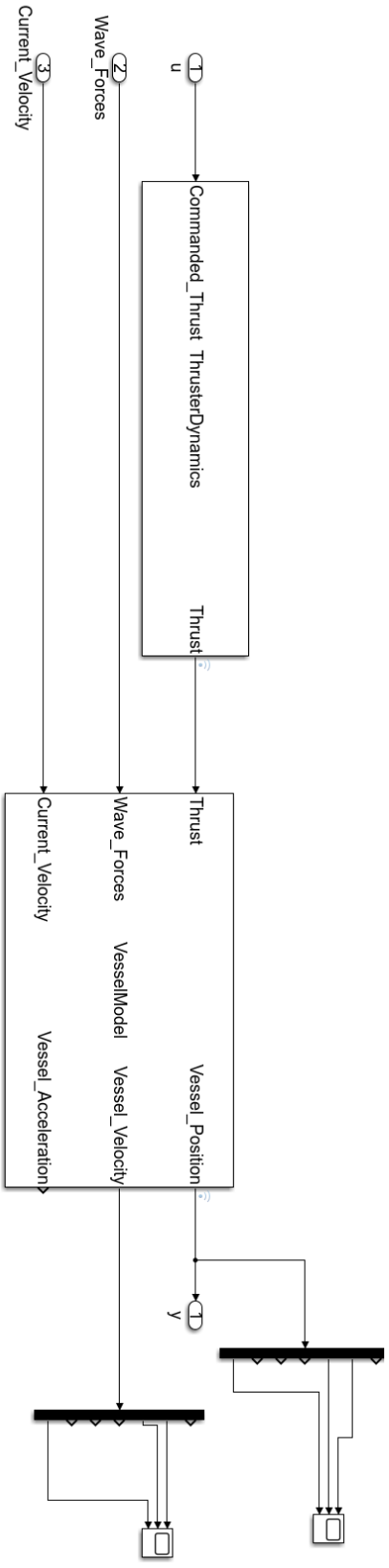
Appendix C – Plotting function for open loop testing.

```
%Fetching the logged data from the MATLAB workspace and insert it into a
%vector.
signal1 = out.Y(:,1);
signal2 = out.Y(:,2);
% Plot signal1 vs. signal2 as an XY diagram
plot(signal1, signal2, 'b-');   %plotting with a blue line
hold on;
% Adds a green marker to indicate the start
plot(signal1(1), signal2(1), 'g>', 'MarkerSize', 10, 'MarkerFaceColor', 'g');
hold on;
% Adds a red marker to indicate the end position
plot(signal1(end), signal2(end), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');
% Add title and labels.
title('North east diagram');
ylabel('North');
xlabel('East');
grid on; % Adds grid to the plot for better readability
```

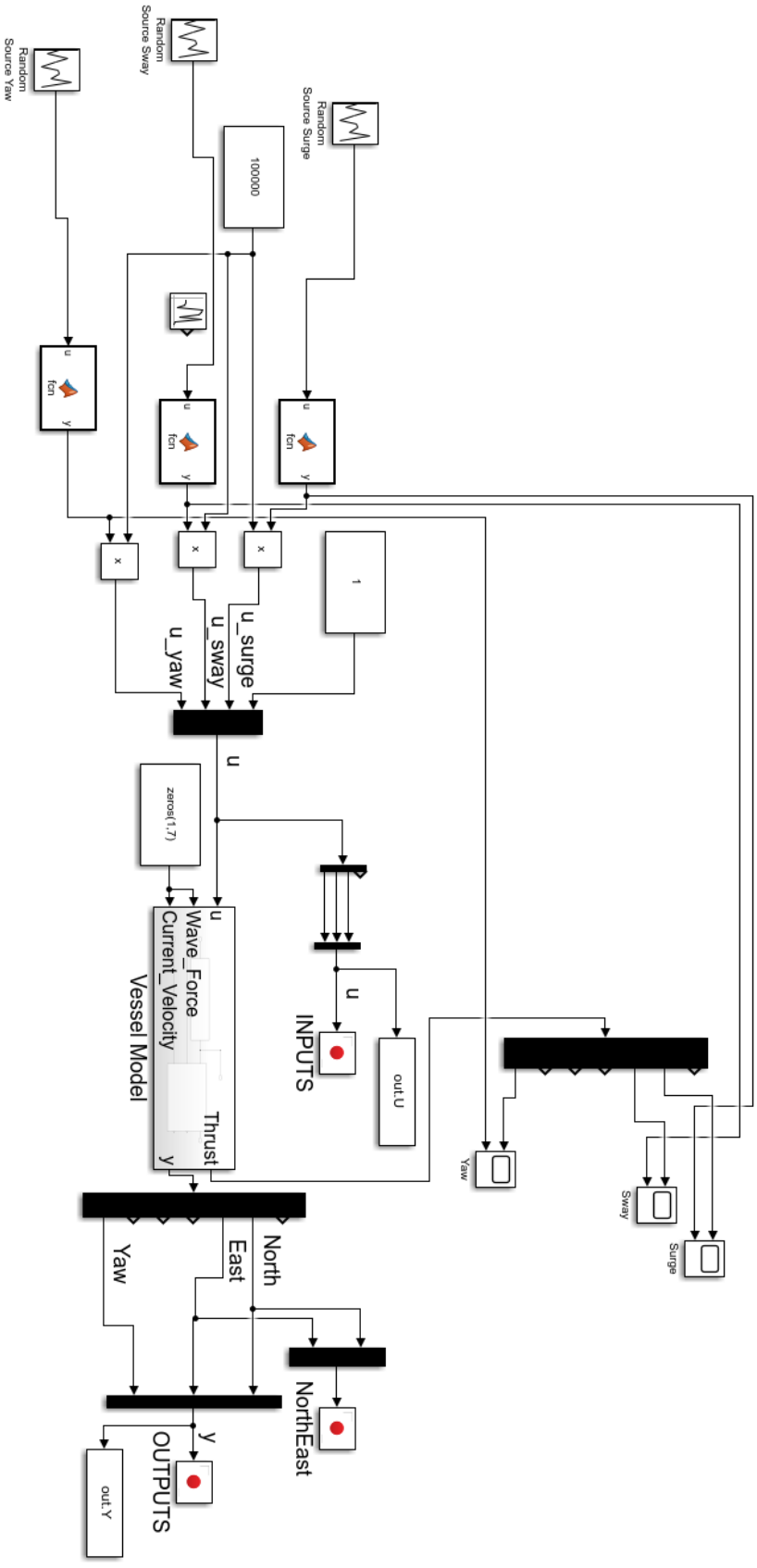
Appendix D – Open loop testing.



Appendix E – Open loop testing Vessel Model.



Appendix F – System Identification.



Appendix G – System Identification control input function.

```
function y = fcn(u)

if (u> 0)
    y=1;

else
    y=-1;
end

```

A function to make the random source a square wave function.

Appendix H – System Identification input and output matrixes.

```
%Inputs
SurgeInput = out.U(:,1);
SwayInput = out.U(:,2);
YawInput = out.U(:,3);

%Outputs
positionNorth = out.Y(:,1);
positionEast = out.Y(:,2);
positionYaw= out.Y(:,3);


%Check that all vectors are of same length
if length(SurgeInput)==length(SwayInput) && length(SurgeInput)==length(YawInput) ...
        && length(SurgeInput)==length(positionNorth) && ...
        length(SurgeInput)==length(positionEast) && ...
        length(SurgeInput)==length(positionYaw)


    %Making input matrix
    U = [SurgeInput, SwayInput, YawInput];
    %Making output matrix
    Y = [positionNorth, positionEast, positionYaw];

else

    error("Data vectors are of different lenght");

end
```

Appendix I – System Identification D-SR.

```
3
4       %D-SR based om input and output matrices.
5       L=2;
6       [A,B,C,D,CF,F,x0]=dsr(Y,U,L);
7       K=CF*inv(F);
```

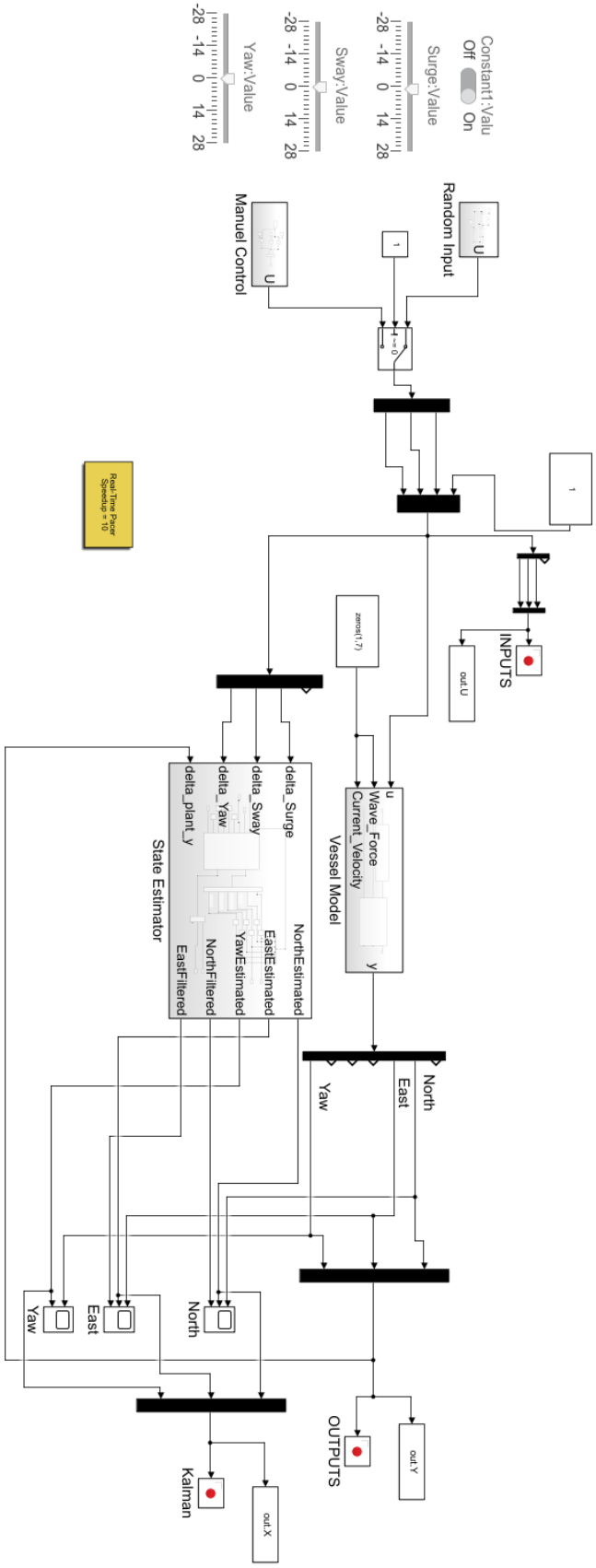
Appendix J – Kalman Initializing.

```
%Kalman initializing

A =[1.0034,0.0207,0.4737,0.0463,-0.2364,2.3069;...
   -0.0011,0.9997,-0.0541,0.1380,0.9540,-13.3772;...
    0.0000,0.0000,0.9424,0.6183,-0.2177,2.1255;...
   -0.0000,-0.0000,0.1676,-0.8000,0.4213,-5.1737;...
    0.0000,-0.0002,-0.0060,0.1014,-0.1223,-12.4611;...
   -0.0000,0.0000,-0.0000,-0.0004,-0.0028,-0.5637];

B = 1.0e-08 *[-0.8484,-0.3258,-0.1868;...
    0.4781,-0.1277,-0.2667;...
   -0.0948,-0.0255,-0.0254;...
    0.2341,0.0767,0.1066;...
   -0.0197,-0.0031,-0.1006;...
   -0.0005,-0.0006,0.0006];

C =[0.6830,-0.1854,-0.2271,0.6690,0.0090,0.0006;
    0.1784,0.6819,-0.0555,-0.0024,-0.7072,0.0085;
   -0.0048,-0.0269,-0.6671,-0.2302,0.0345,0.7071];


 D =1.0e-08 *[-0.3458,-0.1652,-0.0717;...
    -0.0376,-0.0129,-0.1373;...
     0.0011,0.0020,-0.0003];

 CF =[0.0262,0.0068,0.0001;...
    -0.0201,0.0256,0.0002;...
     0.0036,-0.0004,-0.0004;...
    -0.0095,-0.0001,-0.0001;...
    -0.0044,0.0088,0.0002;...
     0.0000,-0.0000,-0.0001];

 F =1.0e-03 *[0.7813,0,0;...
    -0.1125,0.6927,0;...
     0.0020,-0.0019,0.0111];


x0 =[-0.0031;
    0.0010;
   -0.0005;
    0.0014;
   -0.0001;
   -0.0000];

K=CF*inv(F);

K =[34.8690,9.8057,11.2373;...
  -20.4980,36.9942,21.6928;...
    4.6280,-0.6726,-33.1105;...
  -12.2170,-0.1861,-5.2135;...
   -3.8993,12.7559,16.9157;...
    0.0335,-0.0487,-11.0087];
```

Appendix K – Plotting real model and estimated model.

```
%RealModel
signal1 = out.Y(:,1);
signal2 = out.Y(:,2);
%KalmanFilter
signal1Kalman = out.X(:,1);
signal2Kalman = out.X(:,2);


% Plot signal1 vs. signal2 as an XY diagram
plot(signal1, signal2, 'b-');   %plotting with a blue line
hold on;
plot(signal1Kalman, signal2Kalman, 'r-');   %plotting with a red line
hold on;
% Adds a green marker at the start
plot(signal1(1), signal2(1), 'g>', 'MarkerSize', 10, 'MarkerFaceColor', 'g');
% Adds a green marker at the start
plot(signal1Kalman(1), signal2Kalman(1), 'g>', 'MarkerSize', 10, 'MarkerFaceColor', 'g');
hold on;
% Adds a red marker at the end
plot(signal1(end), signal2(end), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'r');
% Adds a red marker at the end
plot(signal1Kalman(end), signal2Kalman(end), 'ro', 'MarkerSize', 10, 'MarkerFaceColor', 'b');
title('North east diagram');
ylabel('North');
xlabel('East');
grid on; % Adds a grid to the plot for better readability
legend('Real model','Estimated values','Start position Real model',...
    'Start position estimated model',...
    'End Position real model','End Position estimated model')
```

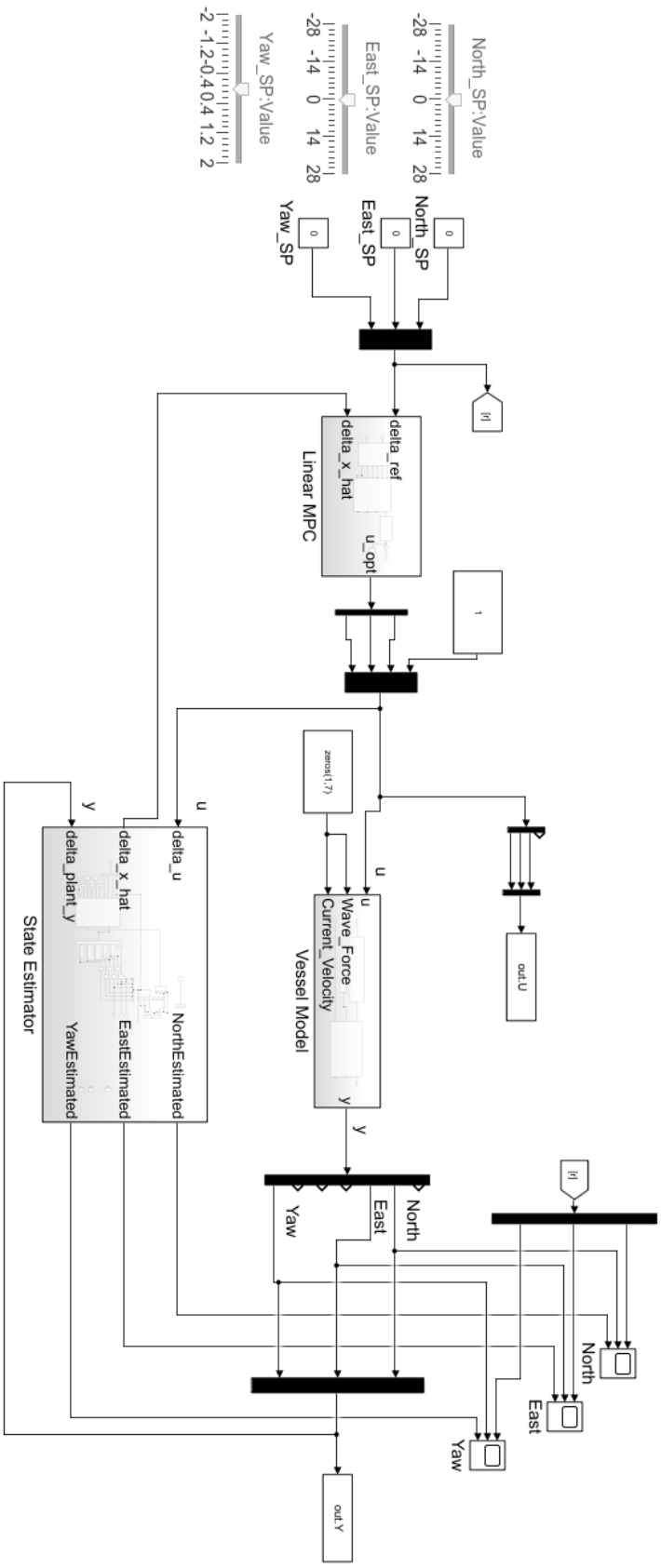
Appendix L – Real model and the state estimator.



Appendix M – QP formulation script.

```
function [H, c, Ae, zL, zU, be] = QP_formulation(delta_ref, delta_x_hat,A, B, C)

%Discrete time model is imported from
%The system identification part.

N = 20;                          %Prediction horizon length
Q = diag([3e4 5e4 1e4 ]);        %Weighting matrix for the error
%%Q = diag([1 1 1 ]);       %Weighting matrix for the error
%P = 1e-3;                       % Weighting matrix for the input
P = diag([2e-6 2e-6 1e-6]);


x0=delta_x_hat;


r=[ones(1, N).*delta_ref(1);
    ones(1,N).*delta_ref(2);
    ones(1,N).*delta_ref(3)];  %make reference for the whole prediction lenght

%Build the QP problem
%Standard QP formulation
nx = size(A, 1); nu = size(B, 2); ny=size(C, 1); nz = N*(nx+nu+2*ny);

% Building H matrices
H11 = kron(eye(N),P);
H22= zeros(N*nx, N*nx);
H33 = kron(eye(N),Q);
H44 = zeros(N*ny, N*ny);
H_mat = blkdiag(H11,H22,H33,H44);

%qpOASES does not support matrices, but only vectors. Stacking collumn wise
H = H_mat(:);
c= zeros(nz, 1);

%Making the constraints
Ae1u = -kron(eye(N),B);
Ae1x = eye(N*nx)-kron(diag(ones(N-abs(-1),1),-1),A);
Ae1e = zeros(N*nx,N*ny);
Ae1y = zeros(N*nx,N*ny);
be1  =[A*x0;zeros((N-1)*nx,1)];

Ae2u = zeros(N*ny,N*nu);
Ae2x = -kron(eye(N),C);
Ae2e = zeros(N*ny,N*ny);
Ae2y = eye(N*ny);
be2 = zeros(N*ny, 1);

Ae3u = zeros(N*ny,N*nu);
Ae3x = zeros(N*ny,N*nx);
Ae3e = eye(N*ny);
Ae3y = eye(N*ny);
be3 = reshape(r, N*ny,1);

Ae_mat =[Ae1u Ae1x Ae1e Ae1y;
    Ae2u Ae2x Ae2e Ae2y;
    Ae3u Ae3x Ae3e Ae3y];

%qpOASES does not support matrices, but only vectors. Stacking collumn wise
Ae=Ae_mat(:);
be=[be1; be2; be3];

%bounds to be inserted when testing is complete
%Building bounds

zL = (-inf*ones(nz,1));
zU= (inf*ones(nz,1));
```

Appendix N – MPC controller



Appendix O – MPC controller with integral action