MAS409, MAS411

Semester Project

Group 8

Active Heave Compensation of Drawwork

Tarjei Skotterud, Martin Dahlseng Hermansen, Martin Mæland

Supervisors
Muhammad Faisal Aftab
Martin Marie Hubert Choux

University of Agder, 2022
Faculty of Engineering and Science
Department of Engineering sciences

Contents

List of Figures

List of Tables

Report Sectionalisation

1 Introduction

- MAS-409
- MAS-411

2 Project Management

- MAS-411

3 Mechanical System

- MAS-409
- MAS-411

4 Electric Machines and Drives

- MAS-409

5 Control System

- MAS-409
- MAS-411

6 Industrial IT

- MAS-411

7 Discussion

- MAS-409
- MAS-411

8 Conclusion

- MAS-409
- MAS-411

1 Introduction

This report is based on a multidisciplinary project in the courses MAS-409 (Electrical Drives and Machines) and MAS-411 (Industrial IT) on the Master's Programme in Mechatronics at the University of Agder. In MAS-409 the students have been taught the fundamentals of electric machines and electromechanical energy conversion, and used that knowledge to understand electric motor drives as the synthesis of electric machines, power electronics and feedback control. Furthermore the students have been thaugt how to model and simulate a variable-speed electric powertrain including power electronic converter, motor, load and feedback control. In MAS-411 the main learning outcome has been to understand, design and analyse larger industrial IT systems. With that the students have been tutored in approaches of systematic PLC programming according to IEC 61131-3, flowcharts, state-machines, ladder logic (LAD), function block diagrams (FBD), structured text (ST), structured control language (SCL), sequential flow charts (SFC). Siemens TiaPortal have been actively used in the course to apply the theoretical knowledge into the real world.

'Active Heave Compensation of Drawwork' is a project all about putting theoretical knowledge into practical thinking. The projects purpose is to virtually model, simulate and control a electrically actuated mechanical system subjected to dynamic loading. The drawworks main duty is hoisting and lowering of the payload to the seabed, and land the payload as softly as possible. In the duty cycle the platform is subjected to irregular wave-motion that needs to be compensated for (heave compensation). For the particular duty cycle explained the manipulation should be achieved through the use of the active drum shown in Figure 1.1. The active drum is connected to the electrical transmission system shown in Figure 1.2.

The complete system should be able to perform multiple load cases with an positional error smaller than 2cm, and be able to heave compensate the irregular wave motion. A robustness-/stress-test should as well be performed to analyze the robustness of the controller and motor. The simulation of the entire system is to be performed as a Hardware-in-the-Loop (HIL) simulation with an user friendly Human Machine Interface (HMI).

To accomplish a HIL simulation as well as being able to use an HMI the drawwork, motor and payload models should run in real time on a Speedgoat. The control system, HMI and State Machine should run on a Siemens PLC that is communicating with the Speedgoat.

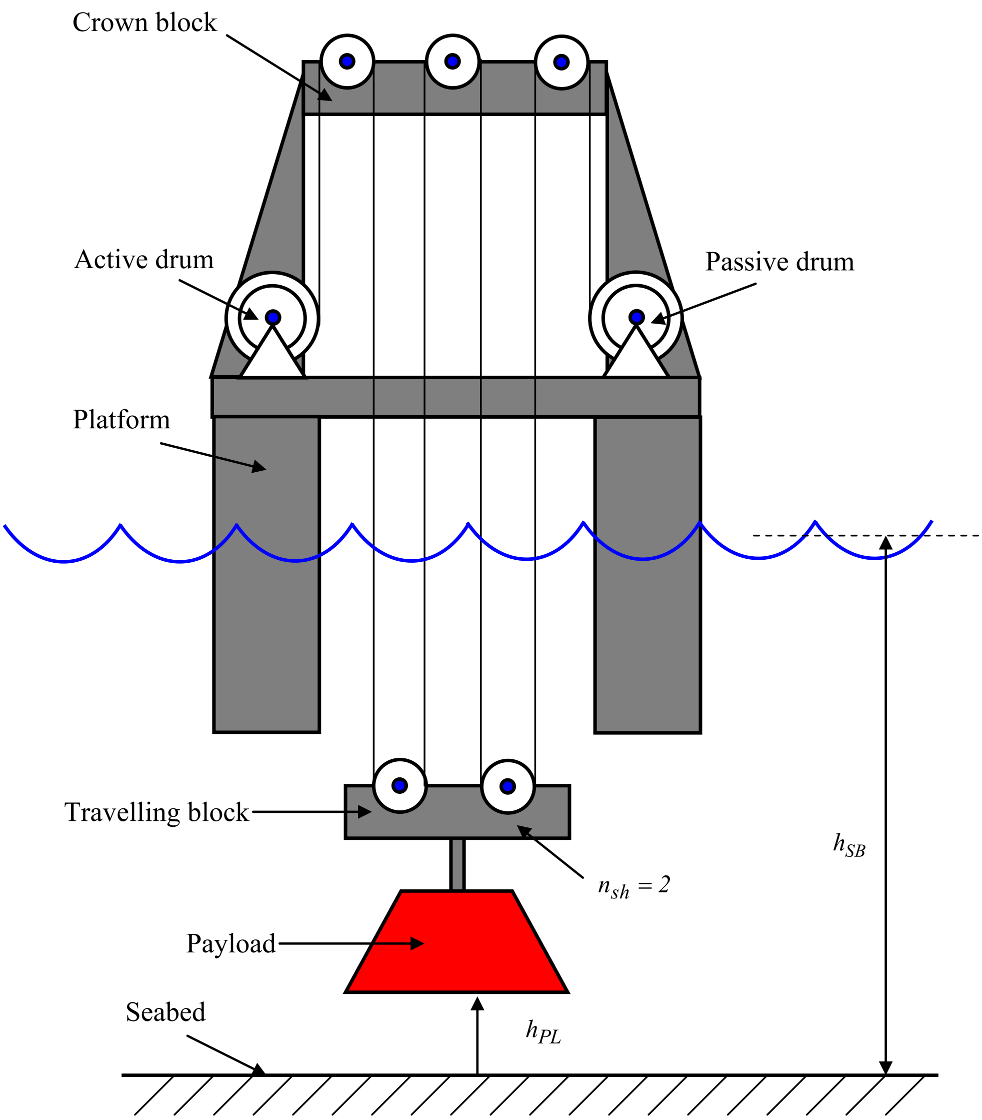


Figure 1.1: The design concept of the drawwork.

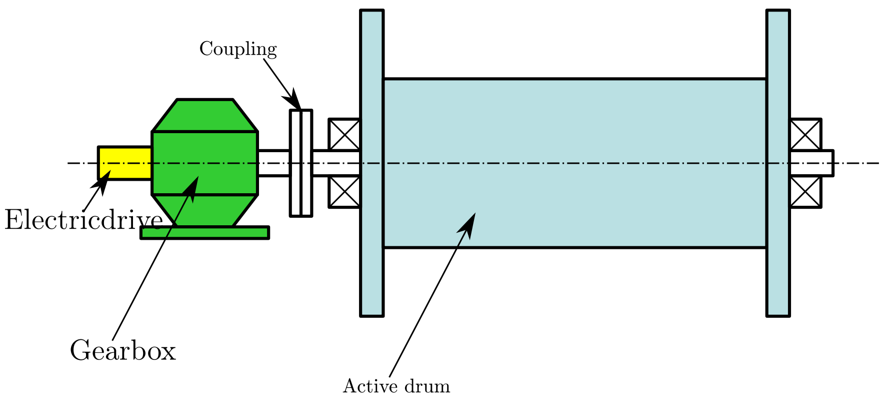


Figure 1.2: Concept for the motor, gearbox and drum.

2 Project management

The project referred to in this report is theoretically and practically wast. Project management has therefore played an important to make sure every member is up to speed, and knows what to do in all stages of the project. At the beginning of the project an overall plan was formed splitting the project in to four main submodules - Manual calculations, selection of components, Matlab Simulink and PLC.

To make sure every member was on top of the project from the beginning all members sat down to discuss and calculate the physical attributes of the system. Then the project were decomposed into the submodules mentioned above. Where each member were set to attend and control a specific part of the project. In that manner every member has an overall oversight for the progress of the specific part.

With the management method mentioned above in mind the group sat down at least once each week for a meeting to discuss the progress of their part, as well as discussing the progress and problems. Every supervised meeting were planned with a agenda, and documented with minutes of meeting, see Appendix A.2. The group varied between being the meeting leader and minutes taker, which meant every member got valuable experience. In the meetings problems was discussed and solved. Furthermore it included any member into all submodules so the entire group was up to speed on the entire path of the project. Towards the end of those meetings it was decided which focus areas should be tagged in the coming week. If a focus area were time essential the whole group would gather to finish the given task as soon as possible. By having weekly meetings the opportunity for each member to multitask, which lead to productivity loss and possibly to no task fully completed, were reduced. During the meetings it was essential to provide a space for an open dialogue. By having an open dialogue each any member is less likely to be left hanging. Meaning a member of the team could fall out because of e.g. not understanding a certain concept. An open dialogue open up for a "no stupid questions" attitude among the team. Additionally it creates a culture that easier can bring up topics such as performance and constructive feedback. By aiming for regular communication and transparency in the meetings the team is more likely to perform better as a unit.

In the beginning of the meeting agenda the work completed that week were listed. Here the minutes of meetings for the former week was added and discussed as well. The minutes of meeting from each week were short and concise to avoid unnecessary information. Most often the minutes of meeting contained notes of questions and important answers from the meeting. Those were worked with during the week, and represented/explained during the next meeting.

2.1 Overview

The project plan has been monitored, evaluated and updated as the project grew. Some specific part of the project has taken longer to finish, and some shorter. However, with using Gantt Chart as a tool the group has had a distinct approach, and felt comfortable reaching the end goal.

A Gantt chart is a useful way of displaying activities such as tasks/events displayed against time. On the left of the chart is a list of the activities, and along the top is the time scale. Each activity is represented by a bar; the position and length of the bar represents the start date, duration and end date of the activity. With this one can easily see what the various activities are, when they begin and end, how long it is scheduled to last, how they overlap and when the entire project starts and ends.

The Gantt chart for this given project can be seen using the link in the footmark[1] or in Appendix A.1.

[1] https://docs.google.com/spreadsheets/d/1ihUSJ1aAfQsOI_CTYKJwMS9ub4hR6Kon-jD3Tl6oTt8/edit?usp=sharing

3 Mechanical System

The simulation model of the mechanical system, also referred to as the drawwork, can mainly be divided into two parts; platform- and load-model. With the latter being dependent on the prior because the platform is subjected to an irregular wave motion which affects the load. A trajectory control signal is also to be feeded into the platform model to manipulate the load, see Figure 3.1. Matlab Simulink is used with the Simscape library to develop the drawwork model.

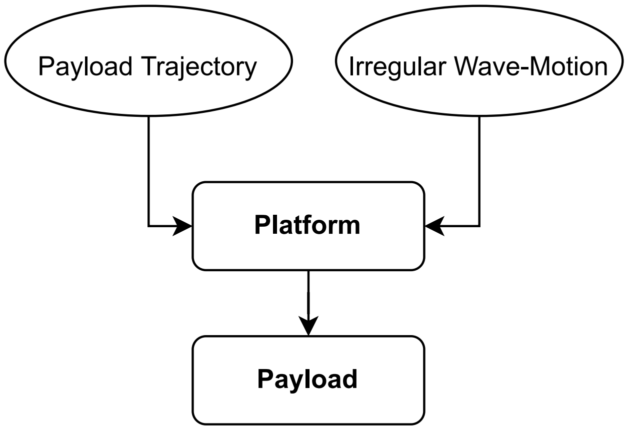


Figure 3.1: Simplified model of drawwork.

3.1 Mechanical System

3.1.1 Platform

The platform model is subjected to an irregular wave motion. As well as it is connected to the load with the five sheaves and two drums shown in Figure 1.1. The sheaves should be modelled as ideal with no friction between them and the cable. The drums are have a diameter of 23.23 cm, has a viscous friction 0.001 $\frac{Ns}{m}$ and a inertia of 1.0 kgm^2.

On the left hand side in Figure 1.1 the active drum is located. That drums hoists and reels the cable because the drum on the right is passive, which means it is basically only connected to the platform. The sheaves and drums together generates a pulley system with an equivalent gear ratio of 4. That means the force on the active drum is reduced by 4, but have to move 4 times as fast to move the payload. Furthermore the drum/wire connection is simplified by using a static radius on the drum. In a real system the drum radius would increase and decrease with the length of the rope.

3.1.2 Payload

The payload is the controlled manipulated part in the system. It hangs below the platform in the sea, and should be placed softly on the seabed. To manipulate the payload correctly there are external forces that needs to be taken into account. Those are gravity, buoyancy, drag and seabed force. See Figures 3.3 and 3.2.

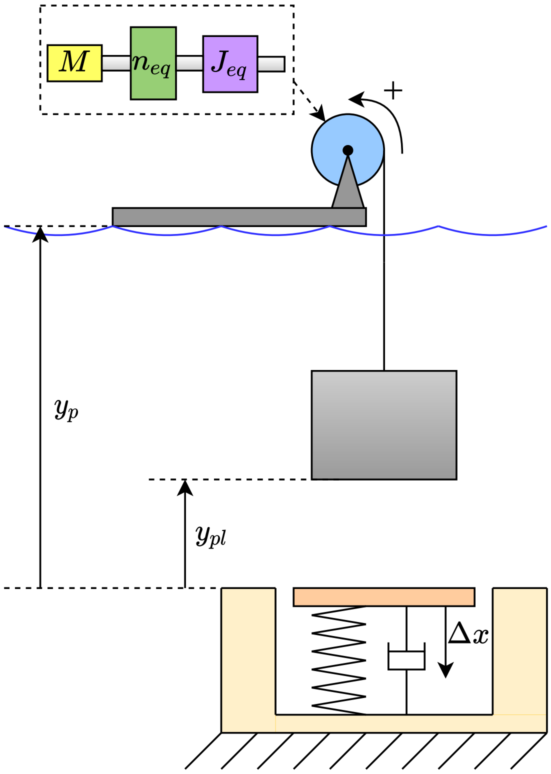


Figure 3.2: Equivalent system

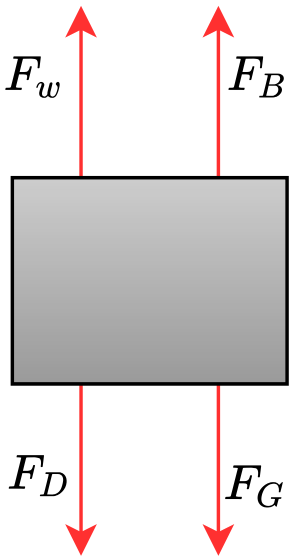


Figure 3.3: Payload FBD

The gravity force applied to the payload can be taken into account by equation 3.1.

$$G = (m_{pl} + m_{tb}) \cdot g \tag{3.1}$$

Notation:		Description:	Value:	Unit:
m_{pl}	-	Payload mass	12000	kg
m_{tb}	-	Traveling block mass	600	kg
g	-	Gravity acceleration constant	9.81	$\frac{m}{s^2}$

The buoyancy force is caused by the payload being soaked in water. That force will counteract with the gravity and try to push the body upwards. It is given by equation 3.2.

$$F_{buo} = \rho \cdot g \cdot V_{pl} \tag{3.2}$$

where

Notation:		Description:	Value:	Unit:
ρ	-	Density of sea-water	1027	$\frac{kg}{m^3}$
V_{pl}	-	Volume of payload	2	m^3
g	-	Gravity acceleration constant	9.81	$\frac{m}{s^2}$

Drag forces is applied to all bodies moving in a material, including water. The force shortly explained slows down any body moving that material, and is increased by the velocity of the moving object. Which means it always works in the opposite direction of which the object is moving. It is calculated in equation 3.3.

$$F_{drag} = \frac{1}{2} \cdot \rho \cdot v_{pl}^2 \cdot C_d \cdot A_{pl} \tag{3.3}$$

where

Notation:		Description:	Value:	Unit:
ρ	-	Density of sea water	1027	$\frac{kg}{m^3}$
v_{pl}	-	Traveling block mass	$\sim$	$\frac{m}{s}$
C_D	-	Sea water drag coefficient	1.8	$\sim$
A_{pl}	-	Projected area of payload	1.5	m^2

The seabed force is working on the payload when it is touching the seabed. To model the force an estimation using a spring-damper is used. If the position of the payload is above seabed the force is equal to zero. Else it is working and pushing the payload upwards in Y-direction.

$$F_{seabed} = \begin{cases} 0 & , \quad \Delta y < 0 \\ k\Delta y + b\Delta\dot{y}\sqrt{\Delta y} & , \quad \Delta y \geq 0 \end{cases} \tag{3.4}$$

where

Notation:		Description:	Value:	Unit:
k	-	Spring coefficient	1.8e6	$\frac{N}{m}$
b	-	Damper coefficient	6.5e5	$\frac{Ns}{m^2}$
Δy	-	Contraction of spring	$\sim$	m
$\Delta\dot{y}$	-	Velocity of payload	$\sim$	m^2

The force to keep the payload not moving in a equilibrium state (wire force) can then be calculated by adding the mentioned forces together:

$$F_{wire} = -(F_{buo} + F_{seabed} - G - F_{drag}) \tag{3.5}$$

Note: Wire force working upwards in Y-direction.
F_{drag} is zero when payload is still.
F_{seabed} is zero when the payload is above the seabed.

3.2 Motion Generators

The projects main objective is to complete three different load cases as accurate and fast as possible, while still keeping the cost to a minimum. The platform is subjected to irregular heave motion from a realistic wave spectrum. Each of the three load cases (LC) involves the same significant height and mean period of the waves. However, the required motion of the payload is different.

For LC1, the payload is fully submerged in the sea at a height y_{pl} above the seabed. The drawwork is then supposed to compensate for the waves in order for the payload to stay stationary at the same height with smallest error possible. During LC2, the payload is to be lowered as fast as possible to a lower y_{pl}. Finally, in LC3, the payload should be lowered and landed as gently as possible onto the seabed, meaning minimal impact between seabed and payload. Table 3.1 describes the requirements for each LC.

Table 3.1: Drawwork load cases

Load case	H_s $[m]$	T_w $[s]$	$y_{pl,0}$ $[m]$	$y_{pl,final}$ $[m]$	$y_{p,0}$ $[m]$
1	1.7	10	7.5	7.5	350
2	1.7	10	7.5	5	350
3	1.7	10	5	0	350

3.2.1 Trajectory

In motion technologies, the choice of a suitable trajectory can be crucial in order to maximize the lifetime and operating safety of moving equipment. Big abrupt changes in position will produce an even bigger step in velocity and acceleration. Considering a mechanical system, this type of sudden need of high torques or forces, due to high accelerations and jerk, could be a contributing factor to wear and tear and, in worst case, result in a breakdown. During the design-process of a motion control system, these factors should to be accounted for to ensure smooth and safe operation.

One way to accomplish it is to design the control system such that the controllers itself produces a smooth response to a big step in setpoint. Another method is through trajectory generation with polynomials as a function of time, which is also commonly referred to as *polynomial interpolation.* This method is based on a certain set of constraints to compute the polynomial coefficients, which will describe how the path(s) are traversed with respect to time. The number of necessary constraints is determined by the number of coefficients, thus the order of the polynomial.

$$\mathbf{A} = \begin{bmatrix} 1 & t_s & t_s^2 & t_s^3 & t_s^4 & t_s^5 \\ 1 & t_f & t_f^2 & t_f^3 & t_f^4 & t_f^5 \\ 0 & 1 & 2t_s & 3t_s^2 & 4t_s^3 & 5t_s^4 \\ 0 & 1 & 2t_f & 3t_f^2 & 4t_f^3 & 5t_f^4 \\ 0 & 0 & 2 & 6t_s & 12t_s^2 & 20t_s^3 \\ 0 & 0 & 2 & 6t_f & 12t_f^2 & 20t_f^3 \end{bmatrix}, \quad \underline{b} = \begin{bmatrix} p_s \\ p_f \\ v_s \\ v_f \\ a_s \\ a_f \end{bmatrix} \tag{3.6}$$

where

Notation:		Description:	Value:	Unit:
t_s	-	Initial time	$\sim$	s
t_f	-	Final time	$\sim$	s
p_s	-	Initial position	$\sim$	m
p_f	-	Final position	$\sim$	m
v_s	-	Initial velocity	$\sim$	m/s
v_f	-	Final velocity	$\sim$	m/s
a_s	-	Initial acceleration	$\sim$	m/s^2
a_f	-	Final acceleration	$\sim$	m/s^2

To be able to select constraints for position, velocity and acceleration in both end-points of a trajectory, a polynomial of order five is chosen to account for the six necessary coefficients. A fifth order polynomial to describe position, brings additional benefits in terms of the jerk. Given the fact that jerk is the acceleration differentiated with respect to time, or the third derivative of position, the jerk will be described as a parabolic function instead of a straight line. Equation (3.6) shows the six equations used to describe the time-schedule for each constraint, placed in matrix **A**. The constraints itself will be the right-hand-side of these equations multiplied with the coefficients and is therefore placed in vector $\underline{b}$. Lastly the coefficients is found by solving the system of linear equations:

$$\underline{x} = \mathbf{A}^{-1}\underline{b} \tag{3.7}$$

Which yields the following expression to describe the trajectory position (p) as a function of time. Where x_n represents the n-th entry in the vector of coefficients:

$$p = x_1 + x_2 t + x_3 t^2 + x_4 t^3 + x_5 t^4 + x_6 t^5 \tag{3.8}$$

Equation (3.8) can be time-differentiated to also output velocity and acceleration, depending on what is desired. Figure 3.4 displays an example of a trajectory for LC3 generated with the described method, hereafter referred to as *Quintic Interpolation*. The descending trajectory was generated with $p_s = 5$ and $p_f = 0$ over a period of 10 seconds with end-point velocity and acceleration set to zero.

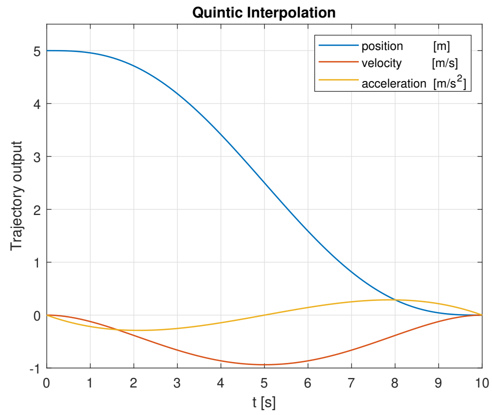


Figure 3.4: $\ddot{y}_p$ noise with discrete and continuous TF

3.2.2 Irregular Waves

The generation of stochastic wave for platform heave motion, is based on the provided Pierson-Moskowitz (PM) example in the project description. The wave elevation is calculated from a Fourier series analysis as a sum of regular wave components with their own frequency, amplitude and phase. Simplistically the stochastic waves elevation can be seen as a superposition of a series of randomly generated sine waves. The equations and theory for the PM spectrum was also provided, but is repeated in this report as a direct citation for continuity.

$$S(\omega) = \frac{A}{\omega^5} \cdot e^{-\frac{B}{\omega^4}} \tag{3.9}$$

$$A = 0.11 \cdot H_s^2 \cdot \omega_1^4 \tag{3.10}$$

$$B = 0.44 \cdot \omega_1^4 \tag{3.11}$$

where

Notation:		Description:	Value:	Unit:
S	-	Spectrum	$\sim$	$m^2 s$
T_w	-	Mean period	$\sim$	s
ω_1	-	Mean frequency	$\frac{2\pi}{T_w}$	rad/s
ω	-	Angular frequency of a single wave	$\sim$	rad/s
H_s	-	Significant wave height	$\sim$	m

The irregular sea elevation is generated by dividing the spectrum into N frequency intervals with width $\Delta\omega$. Furthermore, random phases between $[0, 2\pi]$ are added to each wave component, which then are summed up to obtain the sea elevation as a function of time. [1]

$$\zeta_{An}(\omega_n) = \sqrt{2 \cdot S(\omega_n) \cdot \Delta\omega} \tag{3.12}$$

$$\zeta(t) = \sum_{n=1}^{N} \zeta_{An}(\omega_n) \cdot cos(\omega_n \cdot t + \phi_n) \tag{3.13}$$

where

Notation:		Description:	Value:	Unit:
ζ	-	Sea elevation	$\sim$	m
N	-	Number of wave comps.	$\sim$	rad/s
ζ_{An}	-	Amplitude of a single wave comp.	$\sim$	m
ω_n	-	Angular frequency of a single wave comp.	$\sim$	rad/s
ϕ_n	-	Phase angle of a single wave comp.	$\sim$	rad

The provided example of a PM spectrum implementation was created in a way where the elevation was sampled and stored in an array. In order to run the script continuously as a MATLAB-function in simulink to output position, velocity and acceleration of the wave at each step, a new function was created. This function takes the current simulation time as input, then outputs the heave position as well as its first and second derivative as velocity and acceleration of motion, respectively.

Additionally, the example was implemented with a fixed seed on the random number generator (RNG) which caused the same wave to repeat itself after some time. To create a more dynamic generator, the function was made to change seed automatically at a fixed sample time during simulation.

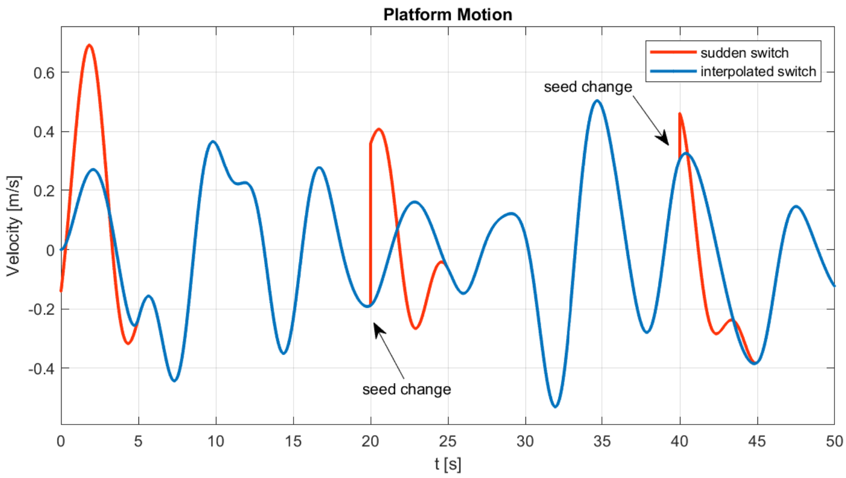


Figure 3.5: Example of the wave generator's velocity output

To ensure a smooth transition between the wave it is currently on and the wave with a need seed, the Quintic interpolation method from 3.2.1 is used during switching. The initial trajectory constraints will be the position, velocity and acceleration of current wave and the end-point constraints are set to the new wave, only T_{sw} seconds ahead in time. When the trajectory finishes, it continues the new wave until next change in RNG seed.

The total time-period to play each wave (T_{seed}) and the switching time (T_{sw}) is both taken as function inputs for simple configuration in Simulink. Figure 3.5 shows an example of the velocity output of the custom wave generator, where $T_{seed} = 20$ and $T_{sw} = 5$. The plot displays the difference between wave switching with- and without Quintic interpolation. Likewise, at the very beginning of the simulation, the wave is interpolated from zero, with the same switching time, to create a more gentle start. The complete MATLAB function is listed in B.1.

3.3 Simulink Model

The equations elaborated earlier in this chapter describes the dynamics of the drawwork. From those equations a simulation model is made to simulate the system. The simulation model is created using Matlab Simulink with the Simscape library. Instead of simulating the entire system just using mathematical equations it is formed using Simscape blocks from the multibody library. However, the inputs to the drawwork is from ordinary Simulink blocks and the mechanical rotational Simscape library.

To start of the platform motion is modelled as a prismatic joint which inputs the wave position. The platform is connected to the drawwork which combine all entities to the same system. The payload motion is affected by the platform motion through the drawwork. In order to hoist or lower the payload the wires of the drawwork is moved through a revolute joint, which acts as the drum. The drum, or revolute joint, is turned by a torque source which comes from the motor. Further the drum is fixed to the platform, and connected to the payload through a wire. The wire is modelled to be ideal (meaning no friction, mass or inertia) and stiff. With the wire being stiff means it basically is conected as a rod between the drum and payload.

The revolute joint is modelled as seen in Figure 3.6 and can be quite difficult to grasp. The input is as seen a torque source, and the angular velocity of the joint is fed back into the torque source with a velocity source in between, which seem odd. However, in order to model the drum with the correct dynamics the torque applied to make the drum rotate needs to be the difference between the torque from the motor and the torque created by the system. If the torque created by the system is larger than the electromagnetical torque from the motor the payload will fall. Opposite the drum will spool in the cable, elevating the payload.

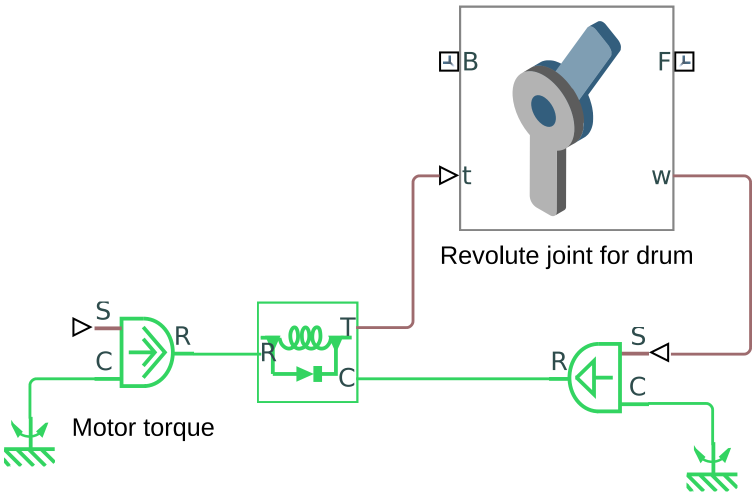


Figure 3.6: Simplified example of revolute drum model.

The force working on the payload is subjected in the same way as the platform wave motion, thus with a prismatic joint. However, here it is force controlled. All the forces working on the payload is combined before being input to the prismatic joint. Additionally the position and velocity on the prismatic joint can be read, those are being used in the drag- and seabed forces evaluated in 3.1.2. The weight of the payload is added as a solid mass to the multibody system. That can be done due to multibody's mechanism configuration which applies mechanical and simulation parameters to the entire multibody machine. Which mean the entire system is set to be subjected by gravity.

In order to simplify the simulation model the sheaves and gear is added to the model at the same time (the gear is further explained and chosen in 4.1.3). They are implemented into the model as one single gear before the drum in the mechanical rotational part of the simulation model.

Instead of implementing the given inertia's in their respective correct locations the effective inertia of the entire system is calculated and implemented just once. To calculate the effective inertia it is essential that each inertia is referred to the location of where the effective inertia is implemented. Which is between the drum and payload in the simulation model.

Since the sheave dynamic is implemented before the drum both the drums and motors inertia needs to be transformed. The latter needs to be transformed through both the gear and sheaves while drum inertia only needs transformation through the sheaves. In equation 3.14 the effective inertia is found.

$$J_{eff} = \frac{J_m}{(i_g \cdot n_{fmsh})^2} + \frac{J_d}{n_{fmsh}} \tag{3.14}$$

where

Notation:		Description:	Value:	Unit:
J_m	-	Inertia of motor	∼	kgm^2
J_d	-	Inertia of drum	1.0	kgm^2
i_g	-	Gear ratio	∼	∼
n_{fmsh}	-	Sheave gear ratio	4	∼

By following the definitions explained in this section the platform- and drawwork simulation model is created. A simplified version is shown in Figure 3.7.

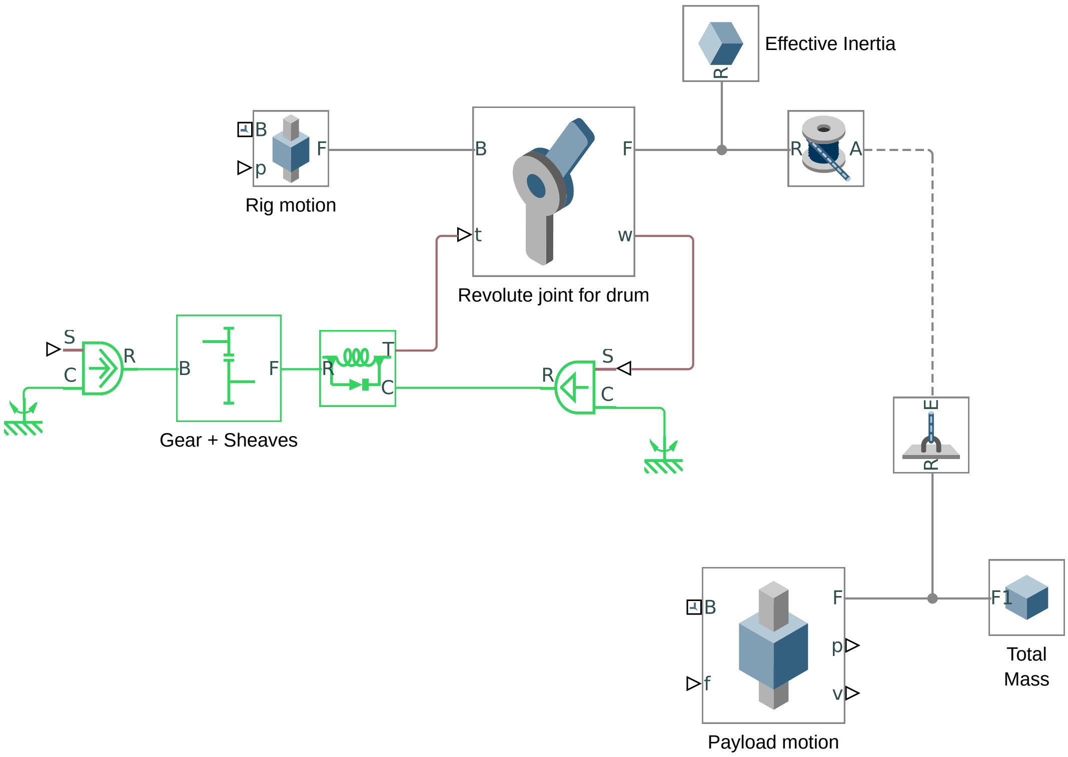


Figure 3.7: Simplified drawwork model.

The entire simulation model of the drawwork can be seen in Appendix E.3.

4 Electric Machines and Drives

4.1 Sizing and Component Selection

An essential part of the project is to size and select some drive train design parameters from catalogs with accuracy, speed and cost optimization in mind. The given components are as well dependent to one another, an important aspect is therefore to conditionally optimize each component to one another.

4.1.1 Sizing

Before starting selection of the components the maximum forces applied to the load given the extreme points of the waves must be found. The extreme point for the wave is where the highest acceleration occurs due to acceleration torque that will occur. One could argue that the motor should be chosen to handle a worst case scenario, e.g. where the trajectory setpoint yield the highest acceleration as well. However, the common work cycle for the motor is to heave compensate for the waves, and the motor should therefore be dimensioned subsequently.

The highest acceleration generated by the waves is found using a Matlab script, see Appendix B.2. 1000 seeds were evaluated, the highest acceleration found is $1.0627[\frac{m}{s^2}]$.

The torque required in the given scenario on the drum is then found:

$$F_{wire,pl} = (m_{pl} + m_{tb}) \cdot \ddot{y}_{wave} + G + sign(yd_{pl}) \cdot F_{drag} - F_{buo} \tag{4.1}$$

$$F_{wire,drum} = \frac{F_{wire,pl}}{n_{sh}} \tag{4.2}$$

$$T_{drum} = F_{wire,drum} \cdot r_{drum} \tag{4.3}$$

Torque required by the motor is also found:

$$T_{friction,drum} = \omega_{drum} \cdot \mu \tag{4.4}$$

$$T_m = J_{eff} \cdot \ddot{\theta}_m + \frac{T_d}{i_g} + \frac{T_{friction,drum}}{i_g} \tag{4.5}$$

4.1.2 Cost Equations

Motor

$$C_M = \omega_M \left(1 + \frac{P_M}{P_{M,max}} + \frac{|n_p - 4|}{4}\right) \tag{4.6}$$

where

Notation:		Description:	Value:	Unit:
p_{pump}	-	Working pressure pump	330	pa
Q_{max}	-	Maximum	200	kW

Drive

$$C_D = \omega_D \left(1 + \frac{P_D}{P_{D,max}}\right) \tag{4.7}$$

where

Notation:		Description:	Value:	Unit:
ω_D	-	~	2	~
$P_{D,max}$	-	Maximum drive power	200	kW

Gearbox

$$C_{GB} = W_{GB} \left(1 + \frac{i_g}{10}\right) \tag{4.8}$$

where

Notation:		Description:	Value:	Unit:
W_{GB}	-	~	2	~
i_g	-	Gearbox ratio	~	~

4.1.3 Component Selection

The selection of drivetrain components is based on an example given in MAS409, where the *Genetic Algorithm* (GA) from the Optimization Toolbox in MATLAB. An algorithm inspired by the process of natural selection. which is, in this case, used as a tool to solve the nonlinear mixed integer optimization problem.

Given in the project is two catalogs from ABB, G.1 and G.2, for motor and drive respectively. The data for 'Flameproof cast iron motors' with efficiency class IE2 are chosen as the range av available motors. They are then converted to .csv format in order for a MATLAB script to be able to import them. The same is done to the 400V wall mounted ACS880-01 drives. The gearbox is considered ideal and is not from any catalog. The project states that a gear ratio between 1 and 10 is available.

Separate cost functions for each component is implemented using the equations shown in 4.1.2. The main cost function that computes the overall cost and normalize it by the mean cost is implemented with optimization for inertia-matching as well.

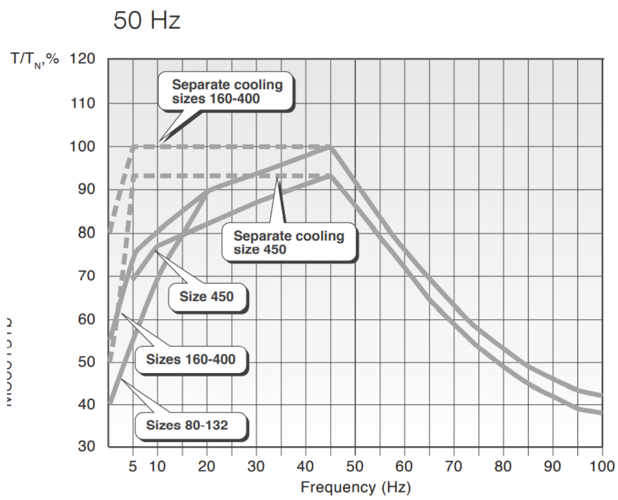


Figure 4.1: Loadability curve from motor catalog

The function in which describes the motors loadability is made using the curve in Figure 4.1, which was found in the motor catalog. With a maximum output power range of up to 200 kW expressed in the project description, the applicable size was assumed to be in the 160-400 region. Since separate cooling was not a part of the cost equation, it was chosen to achieve higher loadability.

Finally is the four constraints that the GA algorithm will use to evaluate if a sampled set of components is adequate. These constraints consists of motor torque, loadability, power and current. The torque constraint is a sum of the continuous load torque and the torque required by motor acceleration.

The continuous load torque is written as:

$$T_{cont} = \frac{F_{w,d} \cdot r_d}{i_g} = \frac{(F_G - F_B)r_d}{i_{sh}i_g} \tag{4.9}$$

Where only the static wire forces at the drum are considered. F_G and F_B is gravity force and buoyancy force respectively. Continuing, the acceleration torque is described with:

$$T_{acc} = \alpha_{max}(J_m + J_L) \tag{4.10}$$

Where

$$\alpha_{max} = \frac{\ddot{y}_p i_{sh} i_g}{r_d} \tag{4.11}$$

and

$$J_L = \frac{J_d + \frac{F_{w,acc} r_d^2}{g}}{i_g^2} \tag{4.12}$$

The wire force due to acceleration is given as:

$$F_{w,acc} = \frac{m_{load} \cdot \ddot{y}_p - F_B + F_G + sign(\dot{y}_{pl}) F_D}{i_{sh}} \tag{4.13}$$

The two torques are then summed together to form the T_{max} constraint. Moving along, an equation for the continuous power is described with:

$$P_{cont} = |\frac{T_{cont} \cdot \dot{y}_p \cdot i_{sh} \cdot i_{gb}}{r_d}| \tag{4.14}$$

Additionally, the total motor current i_m is calculated as a root of each component in dq frame squared. A complete implementation of the drive train selection is available in C.

The GA algorithm converged in the following solution:

- **Motor:** M3JP 315 SMB, 132kW 4P
- **Drive:** 132 kW with $I_N = 246\ A$ and $I_{max} = 350\ A$
- **Gear ratio:** 4.5

The total cost of the components summed together is **10.37** and the result of the added inertia matching desire, J_m/J_{eq}, was about 1.44.

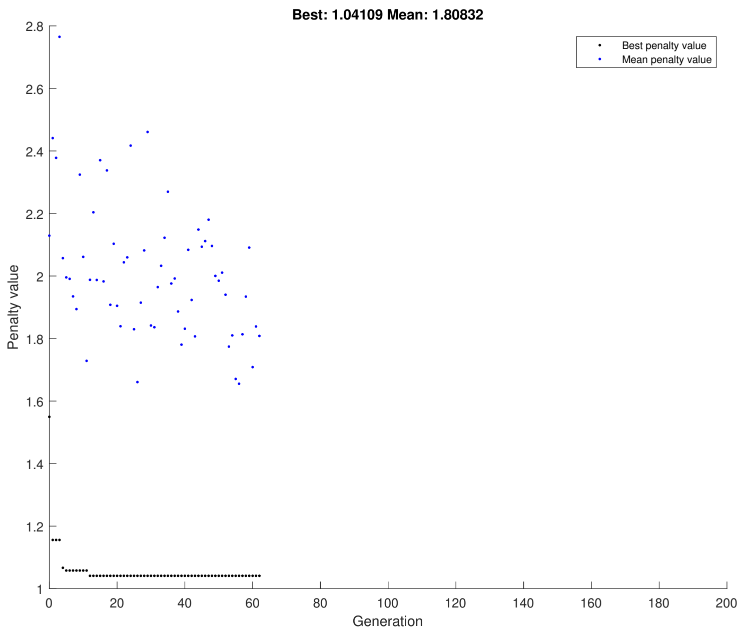


Figure 4.2: Penalty value vs generation

The same script was also tested without the added inertia matching in the cost function, however the GA solver concluded with the exact same result. Figure 4.2 shows the plotted penalty value vs generation.

4.2 Induction Motor Model

Given the motor found in 4.1.3 a dynamic model must be made to simulate it. That is done by estimating the equivalent circuit parameters, and by using dynamic equations to describe the motors behavior. The main elements that have to be included in order for the motor model to behave properly is: the equivalent circuit, torque production and the dynamic flux equations.

The induction motor is magnetized from the stator. Because of asynchronous rotation the current is induced into the rotor winding according to Lenz' law. That is giving forces that counteract the lagging effect which creates the electrical torque. Since the motor is asynchronous the stator and rotors movement is different, the difference is called slip. Due to slip, the rotor flux angle is not readily measurable and has to be estimated.

All principles and theory in this chapter is based on information from lectures in MAS409 and Control of Voltage-Source Converters and Variable-Speed Drives [7].

4.2.1 Dynamic Flux Equations & Equivalent Circuit

Considering the stator circuit the resistance of the stator winding is about equal in all three phases. As mentioned the stator induces current into the rotor. Hence it follows the law of induction. Because of that the part of the stator voltage which is not dissipated in the stator resistance will build up a flux in the stator winding. Therefore, with v_s^s as the stator-voltage space vector, this equation is formed:

$$v_s^s - R_s i_s^s - \frac{d\Psi_s^s}{dt} = 0 \tag{4.15}$$

where

Notation:		Description:	Value:	Unit:
v_s^s	-	Stator voltage space vector	$\sim$	V
R_s	-	Stator winding resistance	$\sim$	Ω
i_s^s	-	Stator current space vector	$\sim$	A
Ψ_s^s	-	Stator winding flux space vector	$\sim$	AH

The rotor circuit can be considered in a similar way. If the rotor is observed with a coordinate system placed on the rotor, splitting it, and the coordinate system is rotating at the same speed as the rotor, there will be no induced voltage because of the rotation. Therefore the rotor-flux dynamics with the coordinates described will be formally identical to the stator-flux dynamics.

$$v_r^r - R_r i_r^r - \frac{d\Psi_r^r}{dt} = 0 \tag{4.16}$$

where

Notation:		Description:	Value:	Unit:
v_r^r	-	Rotor voltage space vector	$\sim$	V
R_r	-	Rotor winding resistance	$\sim$	Ω
i_r^r	-	Rotor current space vector	$\sim$	A
Ψ_r^r	-	Rotor winding flux space vector	$\sim$	AH

Equations 4.15 and 4.16 are noted to different coordinate system, hence superscript s and r. To model a correct induction motor the equations needs to be expressed to the same coordinate system. Which can be done by transforming i_r^r and v_r^r to stationary coordinates. That short-circuits the rotor winding, giving:

$$v_r^r = 0 \tag{4.17}$$
$$i_r^s = e^{j\theta_r} i_r^r \tag{4.18}$$
$$\Psi_r^s = e^{j\theta} \Psi_r^r \tag{4.19}$$

Equation 4.16 is then transformed:

$$0 - R_r e^{-j\theta_r} i_r^s - \frac{d(e^{-j\theta_r}\Psi_r^s)}{dt} = 0 \tag{4.20}$$
$$-R_r e^{-j\theta_r} i_r^s - (-j\omega_r e^{-j\theta_r}\Psi_r^s + e^{-j\theta_r}\frac{d\Psi_r^s}{dt} = 0 \tag{4.21}$$
$$j\omega_r \Psi_r^s - R_r i_r^s - \frac{d\Psi_r^s}{dt} = 0 \tag{4.22}$$

The induction motors dynamic flux equations is therefore described by:

$$\frac{d\Psi_s^s}{dt} = v_s^s - R_s i_s^s \tag{4.23}$$
$$\frac{d\Psi_r^s}{dt} = j\omega_r \Psi_r^s - R_r i_r^s \tag{4.24}$$

By using Faraday's law of inductance the flux can as well be represented using inductances in the circuit. Using this it is possible to find a relation between the flux linkages of the stator and rotor. Presuming the magnetic conditions to be linear the airgap flux can be expressed:

$$\Psi_a^s = L_m i_m^s \quad , \quad i_m^s = i_s^s + i_r^s \tag{4.25}$$

where

Notation:		Description:	Value:	Unit:
L_m	-	Mutual inductance between stator and rotor	$\sim$	H
i_m^s	-	Magnetizing current	$\sim$	A

The sum of the airgap flux and leakage flux yields the total flux. The leakage flux, with linear magnetic conditions, is only proportional to the current passing through. That means the leakage fluxes from the stator and rotor can be represented the leakage inductances. Adding that with the airgap flux gives:

$$\Psi_s^s = L_m i_m^s + L_{sl} i_s^s \tag{4.26}$$
$$\Psi_r^s = L_m i_m^s + L_{rl} i_r^s \tag{4.27}$$

where

Notation:		Description:	Value:	Unit:
L_{sl}	-	Stator leakage inductance	$\sim$	H
L_{rl}	-	Rotor leakage inductance	$\sim$	H

Assuming constant inductances equations (4.23-4.24) and (4.26-4.27) can be combined:

$$v_s^s - R_s i_s^s - L_{sl}\frac{di_s^s}{dt} - L_m\frac{di_m^s}{dt} = 0 \tag{4.28}$$

$$j\omega_r \Psi_r^s - R_r i_r^s - L_{rl}\frac{di_r^s}{dt} - L_m\frac{di_m^s}{dt} = 0 \tag{4.29}$$

A more visualized describing of the equations above is displayed in the dynamic T-equivalent circuit in Figure 4.3.

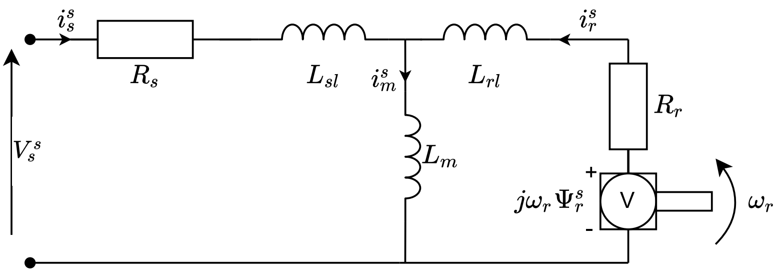


Figure 4.3: Dynamic T-Equivalent Circuit of Squirrel Cage Induction Motor.

The T-equivalent dynamic model is relevant when looking at the physical system. However, when doing analysis and controller design it is less applied due to it being over constrained. The currents in the branches are not linearly independent because $i_m^s = i_s^s + i_r^s$. To simplify one leakage inductance can be used instead. To accomplish this the rotor variables from equation 4.25 can instead be:

$$\Psi_R^s = b\Psi_r^s \quad , \quad i_R^s = \frac{i_r^s}{b} \tag{4.30}$$

where

Notation:		Description:	Value:	Unit:
b	-	Transformation factor	$\frac{L_m}{L_r}$	$\sim$

To further transform the equations found earlier in this chapter new variables can as well be introduced:

$$L_r = L_{rl} + L_m \tag{4.31}$$

$$L_\sigma = (L_{rl}//L_m) + L_{sl} \tag{4.32}$$

$$R_R = \frac{R_r L_m^2}{(L_{rl} + L_m)^2} \tag{4.33}$$

where

Notation:		Description:	Value:	Unit:
L_r	-	Rotor and magnetizing inductance	$\sim$	H
L_σ	-	Total leakage inductance	$\sim$	H
R_R	-	Transformed rotor resistance	$\sim$	Ω

Using the given variables new equations in the dynamic equivalent circuit can be transform the model to a state space form with states $X = [i_s^s, \Psi_r^s]^T$ and input $U = [v_s^s]$:

$$L_\sigma \frac{di_s^s}{dt} = v_s^s - (R_s + R_R)i_s^s - \left(\frac{R_R}{L_m} - \frac{j\omega_r L_m}{L_r}\right)\Psi_r^s \tag{4.34}$$

$$\frac{d\Psi_r^s}{dt} = \frac{R_R L_r}{L_m} i_s^s - \left(\frac{R_R L_r}{L_m^2} - j\omega_r\right)\Psi_r^s \tag{4.35}$$

where

Notation:		Description:	Value:	Unit:
$\frac{d\Psi_r^s}{dt}$	-	Flux emf ≈ rotor emf ≈ back emf	∼	AH

In steady state the rotor emf is in phase with and proportional with the rotor current. That forms the circuit shown in Figure 4.4.

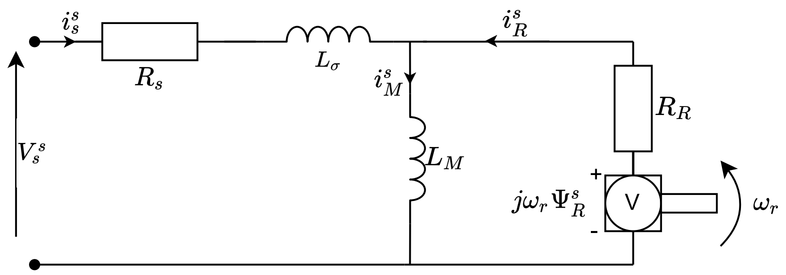


Figure 4.4: Dynamic Inverse-Γ-Equivalent Circuit of Squirrel Cage Induction Motor.

4.2.2 Torque Production

In an equivalent circuit there are active power developed at three locations. Two of which give copper losses, that happens in the stator- and rotor resistances. The power produced by the motor is from the rotor emf, and is described by equation 4.36.

$$P_e = -\frac{2}{3}Re\{j\omega_r\Psi_r^s i_r^s\} \tag{4.36}$$

The negative sign signify that i_r^s is working in the opposite direction to the rotor emf. Working with complex numbers, the active power is also represented by:

$$\begin{aligned} P_e &= \frac{2\omega_r}{3} Im\{\Psi_r^s i_r^{s^*}\} \\ &= \frac{2\omega_r L_m}{3(L_m + L_{rl})} Im\{\Psi_r^{s^*} i_s^s\} \end{aligned} \tag{4.37}$$

This can be applied because the complex value $z = x + jy$ means:

$$Re\{jz\} = e\{j(x + jy)\} = -y = -Im\{z\} \tag{4.38}$$

With the active power found, the electromagnetic torque is calculated:

$$\begin{aligned} T_{em} &= \frac{P_e}{\omega_m} = \frac{pP_e}{2\omega_r} \\ &= \frac{pL_m}{3(L_m + L_{rl})} Im\{\Psi_r^{s^*} i_s^s\} \end{aligned} \tag{4.39}$$

where

Notation:		Description:	Value:	Unit:
p	-	Nr. of poles	4	~

4.2.3 Equivalent Circuit Parameters

Electric motors manufacturers try to keep the technical specifications about their product tight to their chest. It is therefore difficult, if not impossible to dig up the exact electric circuit parameters to a bought electric motor. Estimations of the equivalent circuit parameters must therefore be done to create a realistic simulation model. The estimations is done by running several tests on the given motor. In the industry a motor can be bought, and the test be completed physically. However they were performed using Simulink in this project. The tests to determinate the equivalent circuit parameters, completed in the given order is; DC-test, Locked Rotor Test and No-Load test.

DC-test

To determine the stator resistance R_s a DC-test can be completed. As the name imply, a DC supply is sourced to the stator. In that manner there is none induced voltage in the rotor, and no current will move in the DC. Because of that, the only remaining element used in the equivalent circuit is R_s.

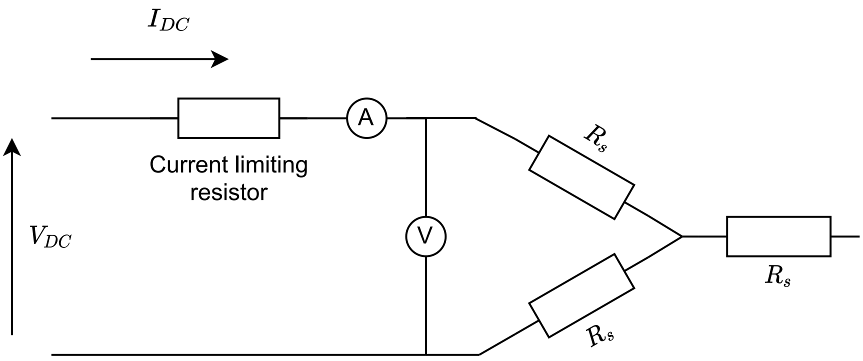


Figure 4.5: Equivalent Circuit of DC-test.

R_s can then be estimated by adjusting the DC voltage source to until the current matches the nominal current given in the motors datasheet. Then R_s can be calculated:

$$R_s = \frac{V_{DC}}{2I_{DC}} \tag{4.40}$$

Locked Rotor Test

A locked rotor test shortly consists of mechanically locking the rotor and running the induction motor to its limit. Since the rotor is locked the value s of the slip will become 1. Which mean that R_r will become much smaller, hence it can be estimated that all current will flow through the rotor. Meaning no current will flow through L_m. The circuitry of the motor can the be seen as a series circuit consisting of R_s, R_r, L_{sl} and L_{rl}.

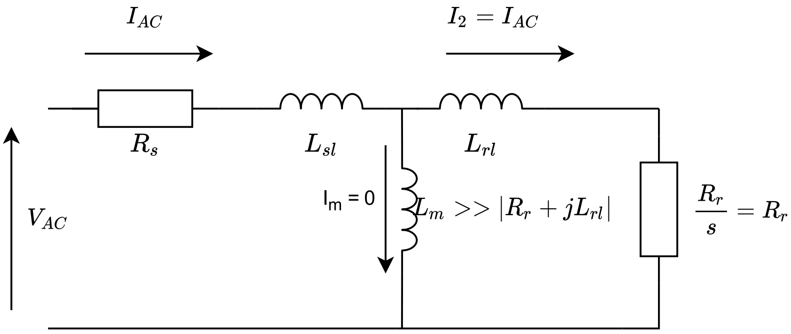


Figure 4.6: Equivalent Circuit of Locked Rotor Test.

To run the motor at maximum a adjustable voltage, adjustable frequency, three-phase power source is adjusted to set the current at full load value. As the motor is run at maximum the line-to-line voltage, line current and total active power is measured. The combined impedance and resistance can then be calculated:

$$cos\phi = \frac{P_{LR}}{\sqrt{3} \cdot V_{LR} \cdot I_{LR}} \tag{4.41}$$

$$Z_{LR} = |\frac{V_{LR}}{\sqrt{3} \cdot I_{LR}}| \tag{4.42}$$

$$|Z_{LR}| = R_{LR} + jL_{LR} \tag{4.43}$$

$$= Z_{LR} \cdot cos\phi + jZ_{LR} \cdot sin\phi \tag{4.44}$$

$$\tag{4.45}$$

Then the remaining resistance and two reactances can be calculated. Since the stator resistance already is known the rotor resistance is found by using the locked rotor resistance and stator resistance. The entire rotor reactance referred to the stator can also be calculated as the reactance is directly proportional to the frequency. The total reactance can therefore be calculated by using the nominal- and locked-rotor test frequency. See equations below.

$$R_{LR} = Z_{LR} \cdot cos\phi \tag{4.46}$$

$$R_{LR} = R_s + R_r \tag{4.47}$$

$$R_r = R_{LR} - R_s \tag{4.48}$$

$$L_{LR} = \frac{f_{rated}}{f_{test}} \cdot L'_{LR} = L_{sl} + L_{rl} \tag{4.49}$$

$$L_{sl} = L_{rl} = \frac{1}{2} \cdot L_{LR} \tag{4.50}$$

Locked Rotor Test

Finally a no-load test can be completed to determine the magnetizing reactance. It consist of applying rated voltage and frequency with no mechanical load attached. Then, as in the locked-rotor test, the line-to-line voltage, line current and total active power is measured.

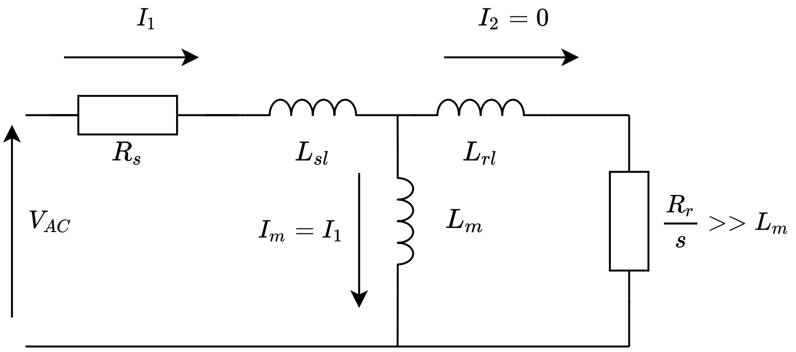


Figure 4.7: Equivalent Circuit of No Load Test.

Since there is no load, and the motor is spinning at rated speed the speed is assumed to be synchronized. The synchronous speed can be achieved by having a slip of 0 which creates infinite resistance in the rotor branch. Because of that it is assumed that all current flows through the magnetizing branch, the rotor branch is therefore neglected.

By measuring the parameters mentioned the magnetizing reactance can be found using the equations below:

$$S_{NL} = V_{NL} \cdot I_{NL} \tag{4.51}$$

$$Q_{NL} = \sqrt{S_{NL}^2 + P_{NL}^2} \tag{4.52}$$

$$L_{NL} = \frac{Q_{NL}}{I_{NL}^2} \tag{4.53}$$

$$L_m = L_{NL} - L_{sl} \tag{4.54}$$

4.3 Field Oriented Control

The motor should be controlled using indirect field oriented control. Which means the reference frame consists of d- and q-direction components contrary to the IM-models $\alpha\beta$-components. A transformation to $\alpha\beta$-components is therefore needed. In order to read the abc-components another transformation is added as well. The transformations can be completed using transformation matrices:

$\alpha\beta$ to dq:

$$v_s = \begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} cos\theta_e & sin\theta_e \\ -sin\theta_e & cos\theta_e \end{bmatrix} \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix}$$

abc to $\alpha\beta$:

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix}$$

dq to $\alpha\beta$:

$$\begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} = \begin{bmatrix} cos\theta_e & -sin\theta_e \\ sin\theta_e & cos\theta_e \end{bmatrix} \begin{bmatrix} v_d \\ v_q \end{bmatrix}$$

$\alpha\beta$ to abc:

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} \frac{2}{3} & 0 \\ -\frac{1}{3} & \frac{1}{\sqrt{3}} \\ -\frac{1}{3} & -\frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix}$$

The reason field oriented control, also known as vector-control, is chosen is because it can operate smoothly around the motors speed range, generate maximum torque at zero speed, and have a high dynamic performance. This can be done due to it depending on the motor parameters to have accurate torque production. Another reason is that in the dq-frame is rather constant contrary to the oscillating of the $\alpha\beta$-frame.

4.3.1 Flux Estimation

In an IM the flux cannot be accurately measured. Because of that it is necessary to estimate it as closely as possible. Here, that is done using the current control model, which is done using the rotor circuit of the motor. To estimate the flux *perfect field orientation* is assumed. By assuming *perfect field orientation* and using dq-coordinates the flux-equation from equation 4.35 is simplified:

$$\frac{d\psi_r}{dt} = \frac{R_R L_r}{L_m} i_d^{ref} - \frac{R_R L_r}{L_m^2} \psi_r \tag{4.55}$$

where

Notation:		Description:	Value:	Unit:
σ_r	-	real flux estimate	$\sim$	AH
i_d^{ref}	-	flux producing current	$\sim$	A

The flux angle (θ_e) used to transform between $\alpha\beta$- and dq-coordinates and the angular velocity of the flux can be found as well:

$$\theta_e = \frac{d}{dt}\left(\omega_r + \frac{R_R L_r}{L_m \psi_r} i_q^{ref}\right) \tag{4.56}$$

$$\dot{\theta}_e = \omega_e = \frac{d}{dt}\left(\omega_r + \frac{R_R L_r}{L_m \psi_r} i_q^{ref}\right) \tag{4.57}$$

where

Notation:		Description:	Value:	Unit:
i_q^{ref}	-	torque producing current	$\sim$	A

4.3.2 Current Controller

In order to use field oriented control on a current controller the stator current equation 4.34 is firstly transformed to synchronous coordinates. This is done using the transformation matrcises mentioned in 4.3. In synchronous coordinates the given equation yields:

$$L_\sigma \frac{di_s}{dt} = v_s - (R_s + R_R + jL_\sigma\omega)i_s + \left(\frac{R_R}{L_m} - \frac{j\omega_r L_m}{L_r}\right)\psi_r \tag{4.58}$$

The term affected by the flux in equation 4.58 is the back emf of the motor, noted E. $R_s + R_R$ is as well re-noted to R_σ:

$$L_\sigma \frac{di_s}{dt} = v' - (R_\sigma + jL_\sigma\omega)i_s + E \tag{4.59}$$

The equation can then be transformed into a Laplace domain open loop transfer function:

$$G(s) = \frac{1}{(s + j\omega)L_\sigma + R_\sigma} \tag{4.60}$$

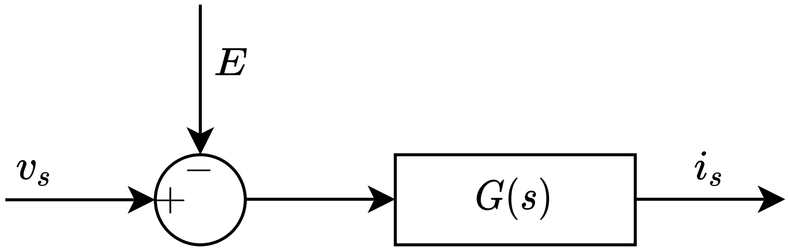


Figure 4.8: Open loop transfer function.

Assuming L_σ is estimated fairly accurate an active resistance, R_a, should be added as an inner loop. By doing this the control error is nearly eliminated without increasing the resistance. [7]

In order to make the inner loop as fast as the closed-loop system the active resistance is calculated by:

$$R_a = \omega_{BW}\hat{L_\sigma} + \hat{R_\sigma} \tag{4.61}$$

where ω_{BW} is the closed-loop system bandwidth with a rise time t_r in millisecond range:

$$\omega_{BW} = \frac{ln(9)}{t_r} \leq 0.04\omega_{samp} \tag{4.62}$$

A decoupling term, $j\omega\hat{L_\sigma}$, can as well be added. The cross-coupling term, $j\omega L_\sigma$, is then removed.

In addition to adding the active resistance and decoupling term a feed-forward of the back emf is added. Those implementations the forms the inner loop:

$$v = v' + (j\omega\hat{L_\sigma})i + \hat{E} \tag{4.63}$$

By implementing the inner loop to equation 4.59 the current controller yields:

$$L_\sigma \frac{di_s}{dt} = v' + (R_\sigma + R_a + j\hat{L_\sigma}\omega)i - (E - \hat{E}) \tag{4.64}$$

$$G'(s) = \frac{1}{sL_\sigma + R_\sigma + R_a} \tag{4.65}$$

The current controller is then complete and closed. See Figure 4.9.

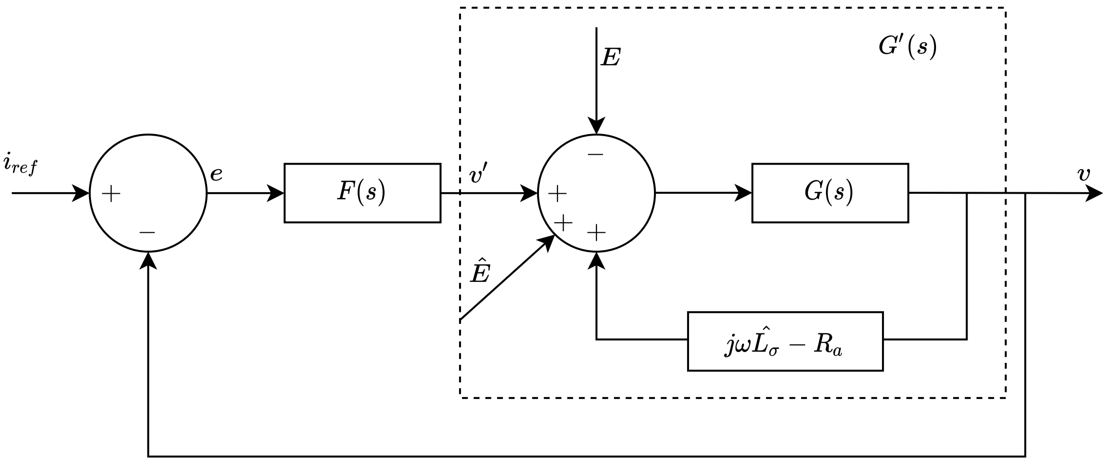


Figure 4.9: Current controller.

Since there are two non-interacting first-order systems, a PI-controller ($F(s)$) is added as well. $F(s)$ being:

$$F(s) = k_p + \frac{k_i}{s} \tag{4.66}$$

By performing block reduction on the closed-loop system,

$$F'(s) = \frac{F(s)G'(s)}{1 + F(s)G'(s)} \tag{4.67}$$

the controller can be designed using direct synthesis because ideally:

$$F'(s) = \frac{\omega_{BW}}{s + \omega_{BW}} \tag{4.68}$$

Hence,

$$F(s)G'(s) = \frac{\omega_{BW}}{s} \tag{4.69}$$

$$k_p + \frac{k_i}{s} = \omega_{BW}\left[L_\sigma + \left(\frac{R_a - R_\sigma}{s}\right)\right] \tag{4.70}$$

Looking at equation 4.70, the proportional- and integral gain can be calculated with the following equations:

$$k_p = \omega_{BW}\hat{L}_\sigma \tag{4.71}$$

$$k_i = \omega_{BW}^2\hat{L}_\sigma \tag{4.72}$$

4.3.3 Field-Weakening

Field-weakening, or in other words flux-weakening, is a technique with the end goal of increasing the operating speed above its limit at expense of reduced torque. Which mean the speed increases with constant power. As known the mechanical power output is torque multiplied with speed. In order to increase the speed without increasing power, the torque is reduced.

To accomplish this the reference signal current to the controller (i_q) is saturated with limits from the nominal-, maximum- and a calculated flux producing current (i_d). See the added block, FW, in Figure 4.10. The re-calculation is done to dynamically adjust the torque- and flux producing currents, relative to each other and the reference. Hence, a field-weakening operation.

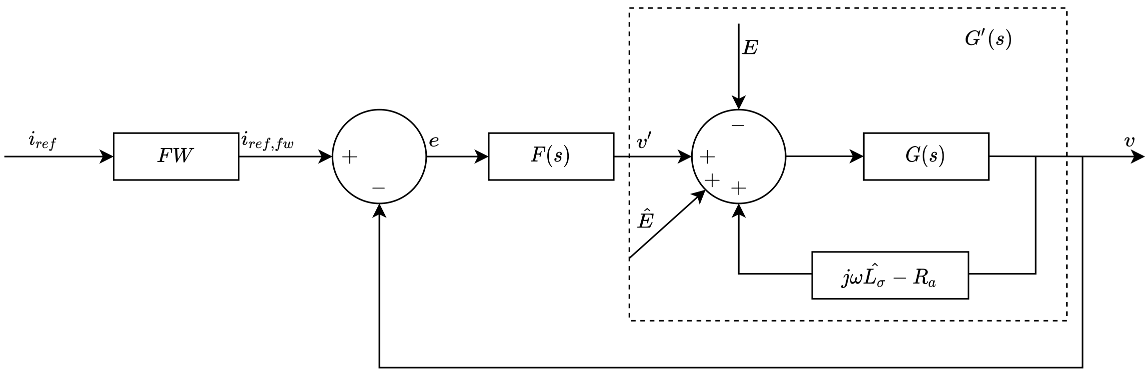


Figure 4.10: Current controller with Field Weakening.

In order to saturate the torque producing current from the flux producing current it firstly needs to be calculated. That is done using ideal feedback voltages in *dq*-domain from the current controller and the angular velocity of the flux found in equation 4.57. In equation 4.73 the flux producing current is found and saturated.

$$i_d^{ref} = k_{fw} \int [V_{base}^2 - (v_d^{ref})^2 - (v_q^{ref})^2] dt \Big|_{I_{min}}^{I_{nom}} \tag{4.73}$$

where

$$\begin{aligned} I_{nom} &= \frac{\psi_{ref}}{\hat{L}_m} \\ I_{min} &= 0.1 I_{nom} \\ k_{fw} &= \frac{\hat{R}_R}{\hat{L}_\sigma^2 \cdot V_{base} \cdot max(|\omega_e|, \omega_{1,N})} \end{aligned}$$

The reason i_d is saturated is mostly to prevent complete demagnetization, however a upper limit is set as well to avoid having a to high flux producing current. By calculating the upper limit from the nominal flux value i_d should never reach overcurrent.

The gain selection of k_{fw} is recommended in [7]. In short terms it is suggested to maintain the closed-loop bandwidth as the gain is decreased as $\frac{1}{|\omega_e}|$ goes above the base speed, but remain constant otherwise.

Contrary to i_d, i_q is saturated dynamically to allow overcurrent in a given time period, also known as the field-weakening range. As seen in equation 4.74 i_q is saturated between its nominal value and the maximum current value of the drive.

$$i_q^{ref} = sat\left(i_{q,nom}^{ref}, \sqrt{I_{max}^2 - (i_d^{ref})^2}\right) \tag{4.74}$$

I_{max} is to be reduced to I_{base} after the maximum time of overcurrent is reached.

4.4 Tests & Results

With all equations to simulate a dynamic induction motor model investigated a Simulink model of the motor were created using standard Simulink blocks to compute the mathematical expressions. The parameters found in this chapter were used to simulate the induction motor model. In order to verify the model several tests is completed giving results on how accurate the model is. The Simulink model is based on models investigated in lectures in MAS409.

To verify the maximum nominal velocity of the IM motor model in nominal working conditions a no load test on the model is done, see Figure 4.11.

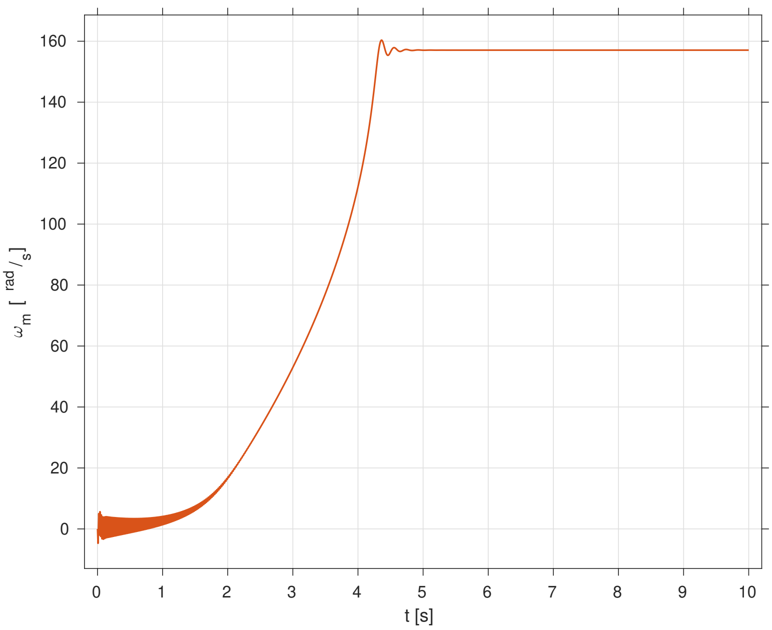


Figure 4.11: Maximum nominal velocity of the IM model.

Tests to verify the output torque is completed. This is done to see what the motors maximum torque is, and how it reacts to a reference signal. In Figure 4.12 a reference signal is sent to the motor. The risetime on the current controller is $1[ms]$ with a step of $0.1[ms]$. In the test the motor has its given inertia from the datasheet, and a constant load of $300[Nm]$ is applied.

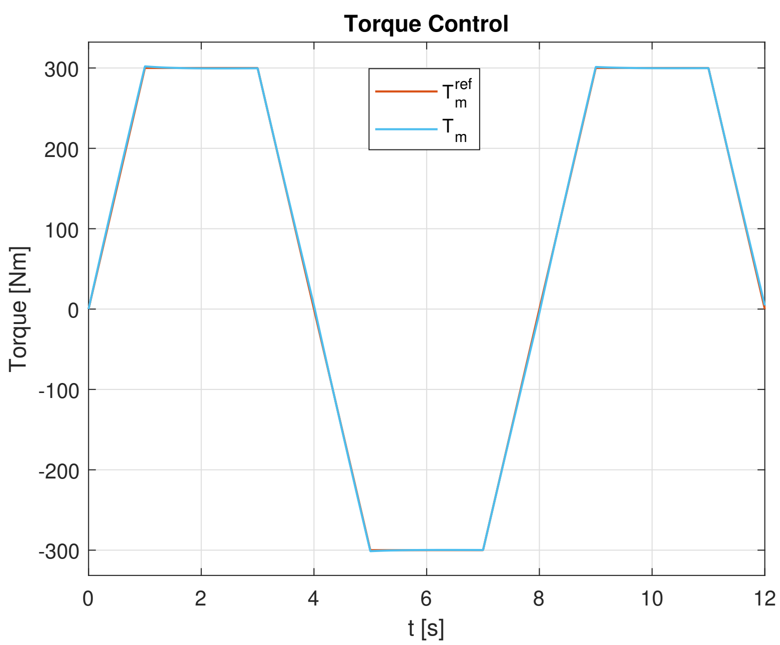


Figure 4.12: Caption

In order to verify that the field-weakening and current saturation is working as intended a setup where the motor has a load applied were created. With this test setup the load and reference can be changed to visually verify the controllers functions:

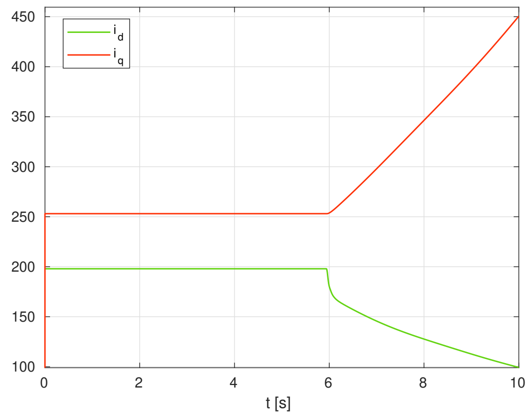


Figure 4.13: dq-current during field-weakening.

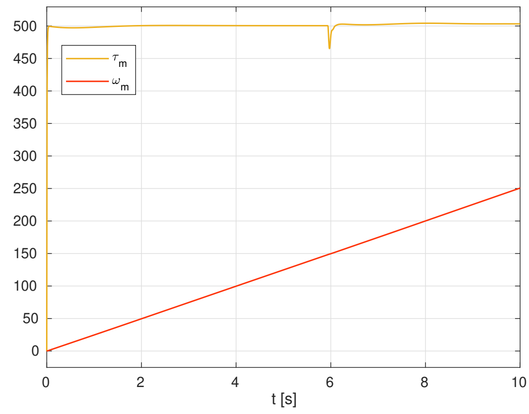


Figure 4.14: τ_m and ω_m during field-weakening.

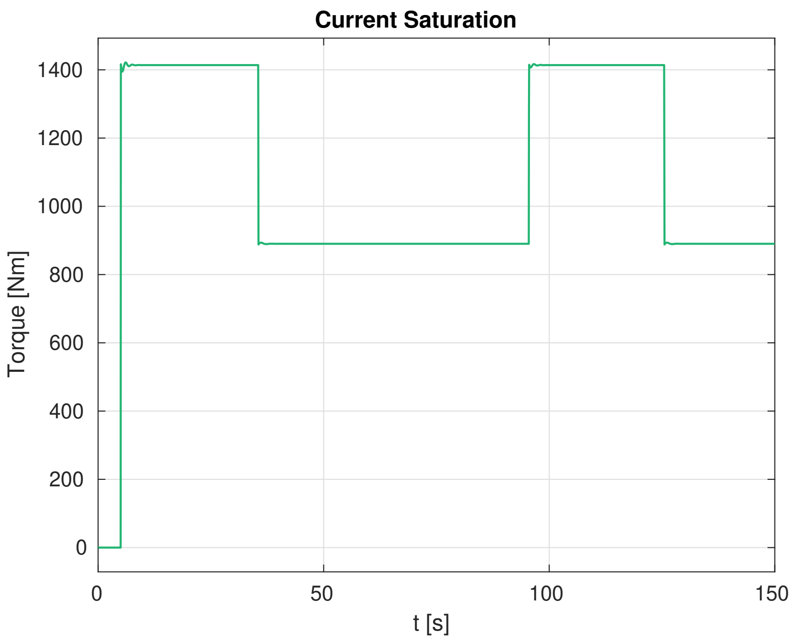


Figure 4.15: Current saturation.

By looking at Figure 4.14 it is seen that when the rotor reaches approximately $150[\frac{rad}{s}$ the torque has a slight dip. At the same time, in Figure 4.13 it can be seen that i_q begins to increase, and i_d starts to decrease. This is what is expected to happen as the motor reaches the field-weakening range as mentioned earlier in 4.3.3. The reason of why the torque is not decreasing is because it is strong enough to continue feeding the reference value. If a higher reference value were chosen, additionally with a higher load, the torque drops when entering field-weakening.

In Figure 4.15 a step reference up to $2000[Nm]$ happens at 5s. The motor is then able to produce about $1400[Nm]$ for 30 seconds. As it reaches the time limit I_{max} is reduced to I_{base} as seen in equation 4.74. When the current it reduced the torque drops. After being at the base current for 60 seconds the motor can enter field-weakening again, the torque is then ramped up again. The time limits are adjusted to better visualize this test.

All Simulink test model setups can be seen in Appendix E.

5 Control System

The project consists of two separate subjects with slightly different requirements and criterias, however the design, implementation and tuning of the control system is done in one single simulink model, created to be directly transferable to the PLC. Furthermore, some necessary configurations and modifications are done on the mechanical plant and drive model in order for them to run on the real-time target with minimal effort. The complete drawwork model together with the induction motor and drive is supposed to run in real-time on target with the PLC as its external controller.

Initially, some design criterias and significant information are set to accomplish the desired performance, as well as maintain the real-time capabilities needed to transfer the designed system.

- During heave-compensation the payload position error has to be less than ±2 cm
- Real-time target step time of 1 ms
- PLC controller sample time of 10 ms
- Be able to control drawwork both with and without heave-compensation activated
- Seabed landing should be gently with minimal impact load between payload and seabed
- Design for a trajectory reference, but also be capable of handling abrupt steps
- Using a systematic method for controller tuning

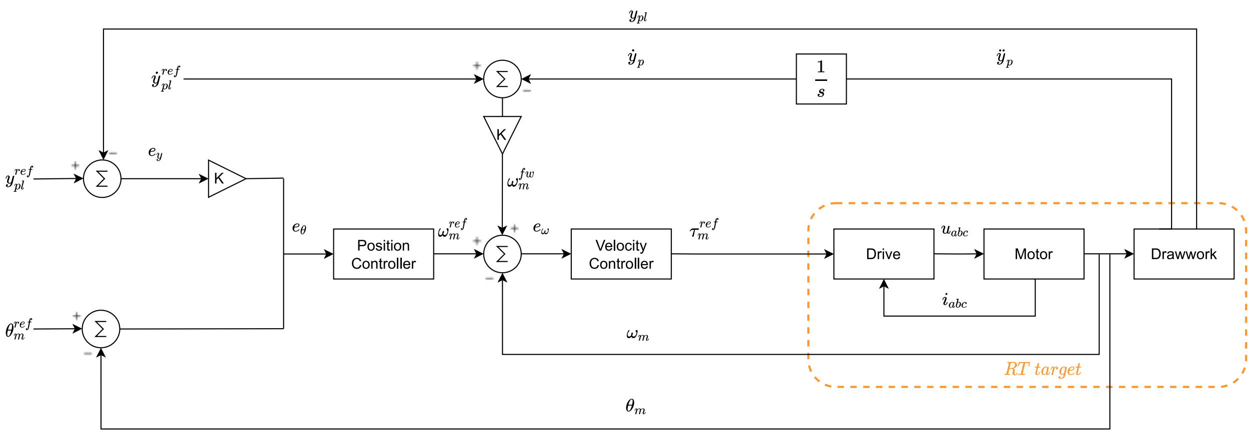


Figure 5.1: Cascaded control-structure overview

The diagram in Figure 5.1 describes the system overview. The control structure consists of three cascaded loops, where the inner motor torque loop is in the drive that was presented in 4.2. The position and velocity controller, including its feedforward, is what is implemented and tuned to later be transferred to the PLC.

Additionally, the position controller is designed to receive an error in angular displacement on the motor. This way it is possible to choose between payload control or direct motor control. The gain, denoted as K, is a conversion factor from kinematic quantities given in the cartesian coordinate-frame, to the equivalent rotational quantities ref. motor. The main reason for such a conversion, instead of relating the two outer controllers to cartesian space, is to be able to use the system without heave-compensation activated. Thus, if heave-compensation is not used, the feedforward loop is manually disabled while the position controller is given an angular displacement as reference.

The conversion factor to equivalent motor quantities is described with:

$$K = \frac{n_{gb} n_{sh}}{r_D} \tag{5.1}$$

where

Notation:		Description:	Value:	Unit:
n_{gb}	-	Gearbox gearing ratio	4.5	$\sim$
n_{sh}	-	Sheaves gearing ratio	4	$\sim$
r_D	-	Drum radius	0.11615	m

Regarding the different controllers, they are initially of type PID. However, the D-term is preferred to disregard due to its negative influence on controller output with noisy signals. Because the D-term calculates the derivative of the error and gains it, it will essentially amplify any noise in the feedback signal which can cause instability. If the D-term is necessary it should be implemented as a filtered derivative, which can limit the D-term to only act on errors below a set frequency.

5.1 Modifications for real-time

Considering the real-time target is supposed to run with *1 ms* step time, both the plant model and the motor including drive has to be discretized and modified. The maximum time of a single cycle has to be below system step time or it will stop the simulation on target.

Through several tests on target, it was revealed that the subsystem for motor with drive was able to run with acceptable performance without changing all the integrals to discrete form. As long as memory blocks was placed to break algebraic loops and rate-transition blocks was set where sample rates are different, the integrals inside the subsystem could remain in continuous form.

Another modification that was necessary to achieve desired response in torque production, was to reduce the bandwidth in the current controller. As described in 4.3.2, the inner loop bandwidth should not exceed 4% of the angular sample rate. Therefore, because the angular sample rate is greatly reduced, a new ω_{BW} was calculated with a closed loop rise time (t_r) of *9 ms* in order to fulfill Equation (4.62).

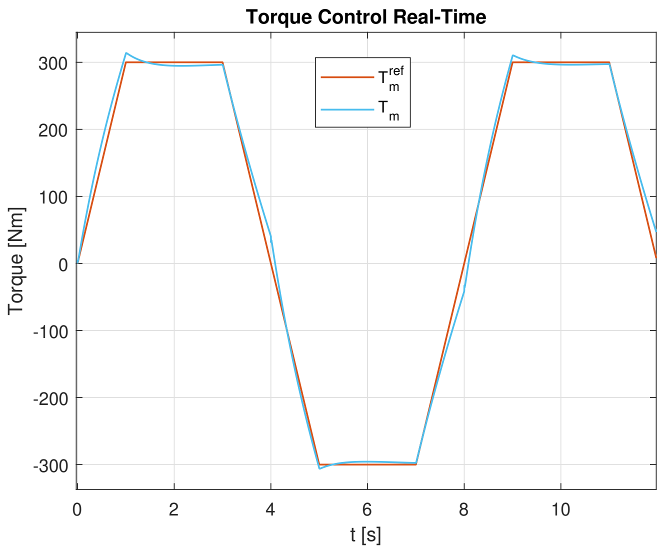


Figure 5.2: Torque response with 1 ms fixed solver step and reduced bandwidth

The new controller gains was tested with the same simulation as the one in Figure 4.12. The graphs from this test is presented in Figure 5.2. There is a slight difference between the two plots, but not more than what was expected, given that the system has a greatly reduced bandwidth while an increase in overall execution step time.

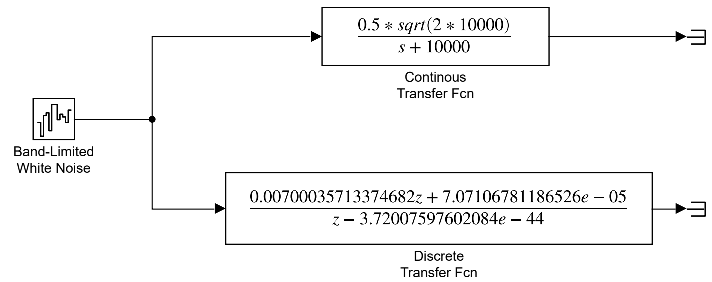


Figure 5.3: $\ddot{y}_p$ noise with equivalent z-transformed TF

Another improvement in real-time performance was the discretization of the noise elements added to the platform vertical acceleration. Initially the band-limited white noise was sent through a continuous filter to produce the noise signal. Investigations revealed that a continuous transfer function running with a relative low fixed sample time, required a lot of computing power. The given filter function was discretized using the built in z-transform function in MATLAB. Figure 5.3 shows the discrete transfer function in parallel with the continuous.

The two transfer functions was then given the same white noise signal to compare outputs. The comparison disclosed that the two filters produce the same output at the required fixed solver step time. A graph of the two outputs is displayed in Figure 5.4.

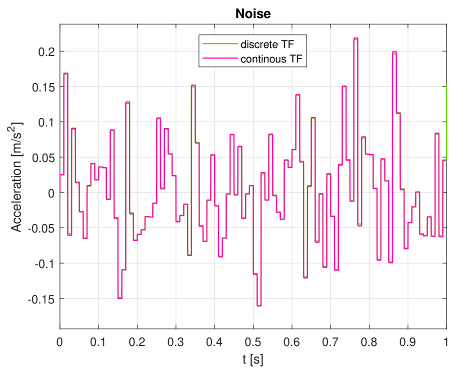


Figure 5.4: $\ddot{y}_p$ noise with discrete and continuous TF

The final modification involves the Simulink and Simscape solver settings. The Simscape settings are configured to use the *Backward Euler* local solver with a fixed sample time of 1 ms. Additonally, the *fixed-cost runtime consistency iterations* box is checked with 10 fixed nonlinear iterations.

On the other hand, the Simulink solver is configured to be the same as the settings used on real-time target. The only setting that is different is that the rate transitions are set to be handled automatically, which is not used in the real-time model. Screenshots of solver configurations in Simulink is found in D.1.

5.2 Velocity Controller

For a cascaded controller to behave properly, the inner (secondary) loop should usually respond to process changes at rates 5-10 times faster than the outer (primary) loop. When the cascaded loops are designed correctly, the output of the primary controller will function as the secondary controller's setpoint. Thus, the secondary controller can react to disturbances that affect the primary loop directly and help reduce the error faster.

The velocity controllers is placed inside the inner loop. This controller is of type PI, given its good reference-tracking capabilities with essentially no steady state error. The D-term is removed due to disadvantages mentioned in the beginning of this section. Furthermore, an integrator without D-term 'damping' may cause some overshoot. However, it was considered that a slight overshoot in velocity may not necessarily cause a problem, as long as it is within reason.

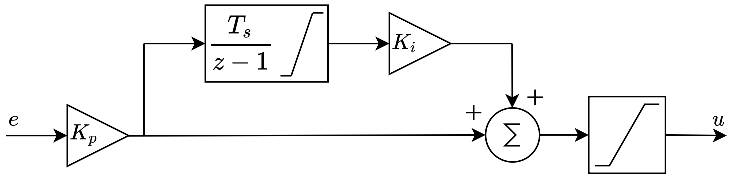


Figure 5.5: Block diagram of discrete PI on ideal form

The PI controller has to be of type discrete and on the ideal form to take advantage of tuning i Simulink and insert the same gains on the PLC. Ideal and parallel form does influence the controller performance itself, rather how each term are gained. On the ideal form, the proportional gain (K_p) contributes to the total gain of the I-term as well. The equation for a discrete PI controller in ideal form is written as:

$$C(z) = K_p \left(1 + K_i \frac{T_s}{z-1}\right) \tag{5.2}$$

Where T_s is the controller sample time. Similarly can the relation between error (e) and controller output (u) be described in continuous time with:

$$u(t) = K_p \left(e(t) + K_i \int_0^t e(t)dt\right) \tag{5.3}$$

A block diagram of the controller is shown in Figure 5.5. However, it has additional saturation on the controller output as well as on the integral, the common anti-windup method known as clamping. The same controller is implemented in Simulink, but with use of the built in Discrete PID block with 10 ms sample time. Output limits is set to the outer region of minimum and maximum deliverable torque, given that the controller output will be the torque reference sent to the electric drive. Although output saturation is not strictly necessary on the velocity controller, given that the drive saturates the current anyway. Regardless, it was done to utilize the built in anti-windup.

5.3 Position Controller with Feedforward

The primary loop consists of a position controller with an additional feedforward. The main task of the primary loop is to keep track of the angular displacement of the motor. Based on the error signal, it will output an angular velocity reference that is sent to the secondary loop to compute the required torque adjustment. The position setpoint for the control system can be given in either payload position or motor position, as long as the primary controller itself receives an angular error ref. motor (e_θ).

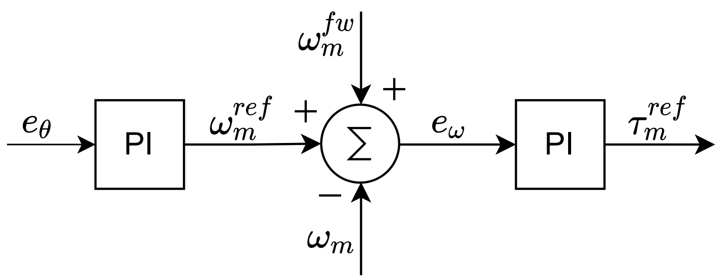


Figure 5.6: Primary and secondary controller with feedforward in between

Initially, it was considered that a simple proportional controller in the primary loop would be adequate, and it was enough to make the error requirement. However, by the addition of an integrator to the position controller as well, the performance increased significantly.

In Figure 5.6, a closer look of the primary and secondary controller is presented. A feedforward signal (ω_m^{fw}) is added to the primary controller output. Given that the output is an angular velocity reference, the feedforward should be the same. This way, with a properly fast and stable velocity controller, the system should be able to control the motor in pure velocity mode, bypassing the primary loop. Pure velocity control is desirable during jog of the payload or drawwork winch directly. Although this feature is an appreciated benefit of the feedforward, it is not the main reason for its inclusion.

Given that the vertical acceleration of the platform ($\ddot{y}_p$) is an available sensor measurement, it can be time-integrated to the equivalent velocity ($\dot{y}_p$) and fed in to the feedforward signal. With this method, it is possible to get significant improvements in heave-compensation accuracy. Simplistically, the control system receives information about disturbances before they are noticeable in the primary loop, which will potentially help to react faster and counteract waves.

Another case where feedforward velocity is helpful, is at times where the payload motion follows a trajectory where its desired velocity is known. This velocity is possible to sum together with platform, then convert in to equivalent motor quantities and add them to feedforward path. Which results in both, enhanced heave-compensation and increased trajectory tracking capabilities, together at once.

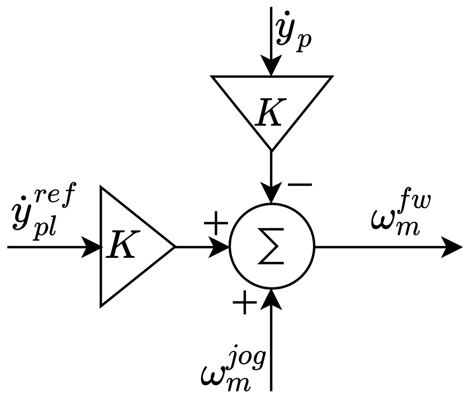


Figure 5.7: Overview of the feedforward paths

The drum has to wind out more wire to compensate for a positive (upward) platform velocity. Hence, the sign of $\dot{y}_p$ is inverted due to the rotational direction of the motor, as shown in Figure 5.7. The gain K, is the conversion factor from Equation (5.1).

5.4 Constant Tension Controller

During seabed landing, the payload will 'sink' a small amount in to the ground because of the spring-damper force working against the payload when it drops below zero. The control system requirements states that the payload should land as gently as possible with minimal impact load. Minimal, or no, impact load essentially means that the spring modeled as the seabed force, should not be compressed more than its length in static equilibrium. Given in other terms, the seabed force should not exceed the equivalent downwards force acting on payload when motionless. The equivalent force is calculated as:

$$F_{pl,eq} = F_G - F_B = (m_{pl} + m_{tb})g - \rho g V \tag{5.4}$$

Where F_G and F_B is the gravity and buoyancy force respectively. This results in a total load of about 103 kN. The value of the spring coefficient (k_s) is a given, thus the compressed change in length can be described with:

$$\Delta x = \frac{F_{pl,eq}}{k_s} \tag{5.5}$$

Due to the size of the spring coefficient, a static equilibrium is present at a compression length of only $\Delta x = 5.7\ cm$. The system has to very gently lower the payload until the acquired spring compression is achieved and stay put. Adding wire-slack to absolutely release payload is unattainable. Which means that the drawwork has to compensate for waves even after a successful landing in order to counteract platform motion and accomplish a stationary payload. One way it could be done is to control the payload position, with AHC activated, to move slowly to the calculated compression length below seabed and keep it at this position. However, this method was considered to be rather unrealistic way to solve the problem, given that the exact position the payload settles at is not necessarily known in a real scenario.

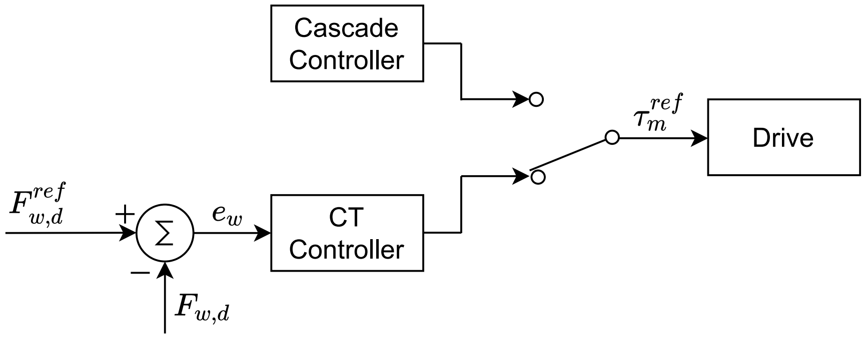


Figure 5.8: CT in parallel with cascade controller

A method that would gain direct control of load impact during and after landing without wire-slack, is a constant tension (CT) controller. The idea behind this controller is to utilize the wire force feedback in a separate loop and control motor torque based on the error between desired and measured tension. Such a system can potentially perform the necessary compensation in order to keep the wire tension at a constant level after landing. To minimize impact load during seabed settling, the controller setpoint can be ramped down until the desired tension level is reached, in which it will be kept constant for as it is needed.

The CT controller is implemented in parallel with the cascade (position and velocity) controller. When the CT controller is active, cascade is inactive and vice versa. Figure 5.8 describes how only one of the two is enabled at once with a switch between them.

5.5 Tuning

The tuning process of a cascade controller requires that the secondary (inner) loop is tuned prior to the primary (outer) loop. The control architecture, that has been derived in this chapter, was implemented together with the mechanical plant and motor with drive model in Simulink. The discrete PID controller blocks was configured to the matching PLC sample time of 10 ms.

The chosen tuning method was the simple experimental method by *Finn Haugen*, named *The Good Gain method for PI(D) controller tuning* [6]. A method in which Haugen says himself is aimed at giving better stability than the Ziegler-Nichols' methods. It is a step by step approach that can be described with the following three steps:

- Initially start with a pure P-controller. Set an initial guess on the proportional gain (e.g. $K_p = 1$) and give the controller a step in setpoint. Adjust K_p until a satisfactory stability is visible in the feedback measurement. The gain should be adjusted until some overshoot with barely observed undershoot is present. This gain is than noted as K_{pGG}.
- The elapsed time between response overshoot and undershoot is then measured. The integral time is set equal to:
$$T_i = 1.5T_{ou} \tag{5.6}$$
Where T_{ou} is the measured time. The relation between gain and time constant is $K_i = \frac{1}{T_i}$
- Reduce K_p some in case the inclusion of the I-term caused some instability. According to Haugen, a good estimate is:
$$K_p = 0.8K_{pGG} \tag{5.7}$$

Following these steps started with the secondary controller with the primary loop disconnected. With the waves turned off, the velocity controller was tuned until acceptable response was reached. Then the primary controller was connected and an initial guess $K_p = 1$ was set. Figure 5.9 displays the step response to this initial guess.

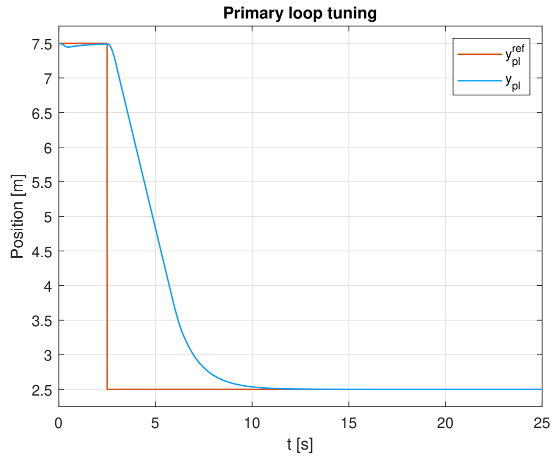


Figure 5.9: Initial guess $K_p = 1$, primary loop

The platform heave-motion was then switched on and further tests where studied. The primary loop K_p was increased until the the controller managed to compensate with errors within the stated criteria of ± 2 cm without being unstable. In order to achieve such a low error without adding the platform velocity as feedforward, an integrator in the primary controller had to be included. At last, the feedforward was included and the position error improved by a significant amount. Figure 5.10 shows how the velocity controller is able to follow its reference signal over a longer period of time.

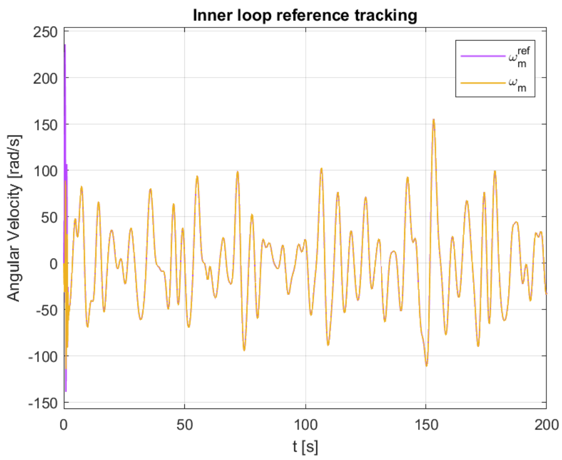


Figure 5.10: Velocity controller heave-compensation

Eventually, the gains for both PI controllers in the cascade system as well as the CT controller was chosen. The primary loop was tuned rather aggressive and its output was therefore saturated to not produce too big velocity references for the even more aggressive inner loop. The saturation limit value is set in order to only have an impact when given a step input. It is checked and confirmed that the output limit does not influence AHC capabilities.

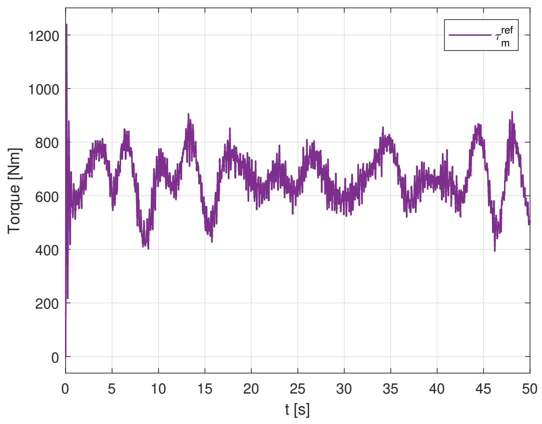


Figure 5.11: Torque reference with unfiltered $\dot{y}_p$

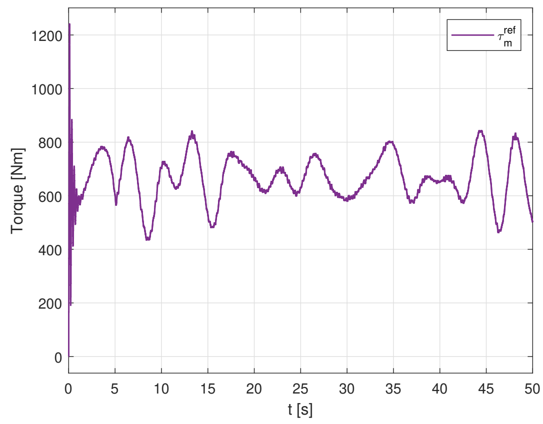


Figure 5.12: Torque reference with filtered $\dot{y}_p$

Further, it was noticed that the white noise in the integrated platform acceleration a lot of noise on the inner loop controller output. Ideally, the inner loop controller should not be so fast that it acts on the noise as well (Figure 5.11). Given that the aggressiveness was a necessary to get minimal error in general, the final solution was to filter the signal a bit. A discrete low-pass filter was implemented after the integral to not cause drift. The filter was configured with 1 ms sample time and with a time constant equal to 0.3. This greatly reduced the noise in output signal (Figure 5.12).

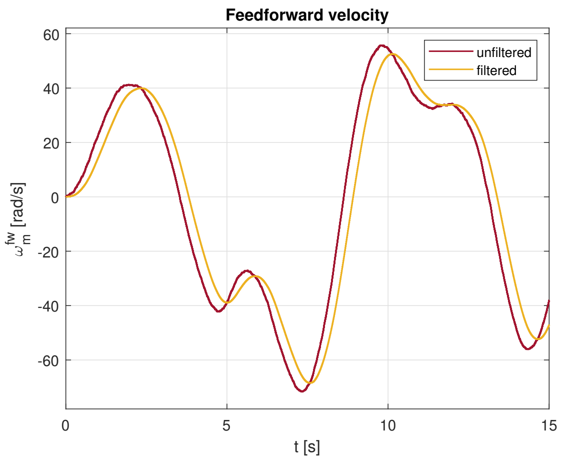


Figure 5.13: Platform feedforward signal comparison

The downside of low-pass filtering is lag, which is seen in Figure 5.13. Even though the low-pass filter increased the position error during AHC, it was better than a very noisy torque reference sent to the drive. The Good Gain method did not result in perfect gains, some adjustment was necessary. The final gains that gave the best performance was:

	K_p	K_i
Primary	15	3
Secondary	88	12
Constant Tension	1.5	3

5.6 Simulation Results

With all the controllers tuned, parameters adjusted and plant models finished in Simulink, a complete simulation for the three load cases was made. This is the finished system implementation for the electric drives part of the project. However, as mentioned earlier this chapter, everything is discretized, tested and tuned to run in real time during HIL simulation.

LC1 states that the drawwork should actively compensate for platform motion and keep payload stationary at a height of 7.5 m above seabed. Perhaps the most important requirement for this load case is the error in payload position. Figure 5.14 shows 20 minutes of active heave-compensation, where a new random wave spectrum is automatically generated every 60 seconds with interpolated switching between each wave is done in 5 seconds.

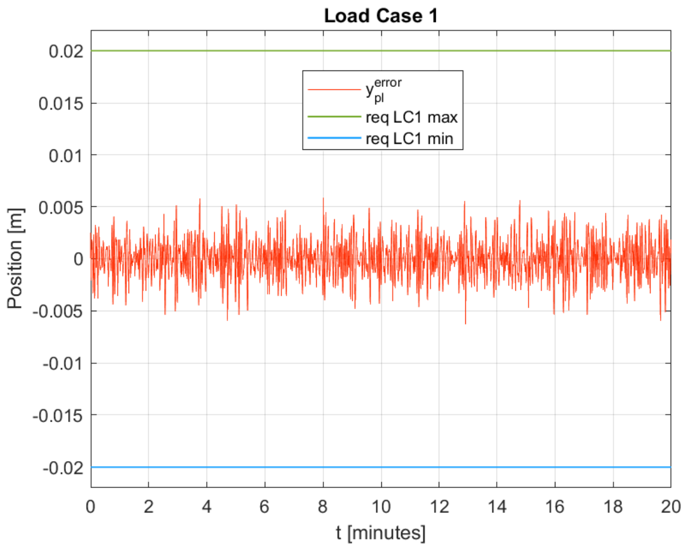


Figure 5.14: Payload position error over 20 minutes of AHC

In addition to the error plot, a plot of the motors angular velocities during the same simulation is displayed in Figure 5.15. Passing through the 60 different wave spectras, the motor stays below field weakening region with a maximum velocity of 1474 RPM.

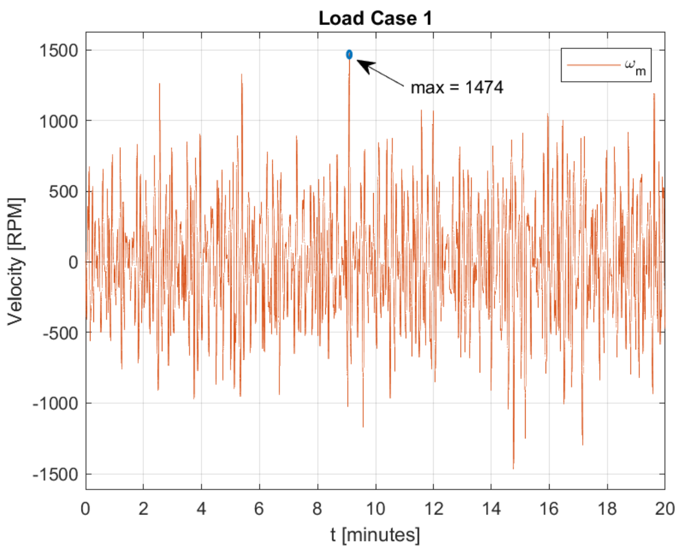


Figure 5.15: Angular velocity of motor during 20 minutes of AHC

For the second load case, the description says that the payload should move as fast as possible from

7.5 m to 5 m. It was not specified if payload was supposed to make a full stop at 5m, nevertheless it was how the case was interpreted by the group. Given that the control system should be able to handle a step and a trajectory input, both had to be simulated in order to find out which one could move though the load case the fastest.

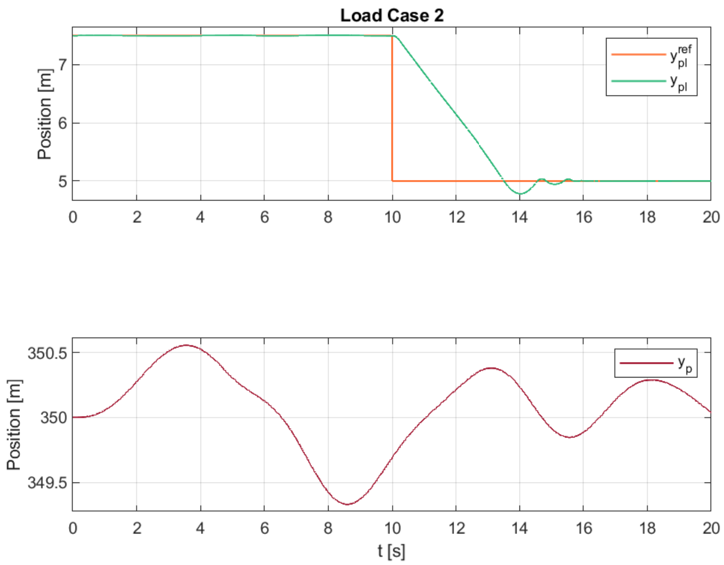


Figure 5.16: Step response in LC2

The time of when the descending step initiates was set to 10 seconds in order to challenge the system more. At 10 seconds, the first wave moves upwards, meaning the winch has to wind out even more wire to lower the payload. A graph of both the step response and the platform position is shown in Figure 5.16.

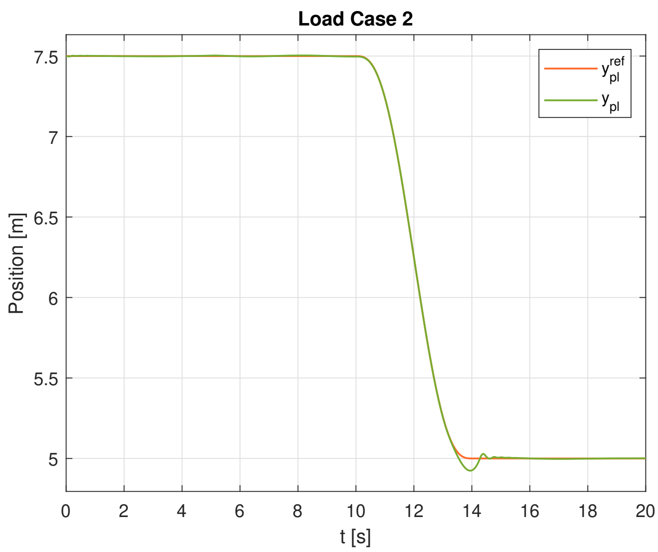


Figure 5.17: Quintic trajectory in LC2

A second test for LC2 was done, but with a quintic interpolated trajectory instead of a step. The result is shown in Figure 5.17. Both the step response and the quintic trajectory presented in the graphs, are done as fast as they possibly can without troubling overshoot. They complete the load case at almost similar time, however the quintic trajectory is a bit faster, overshoots less and settles quicker.

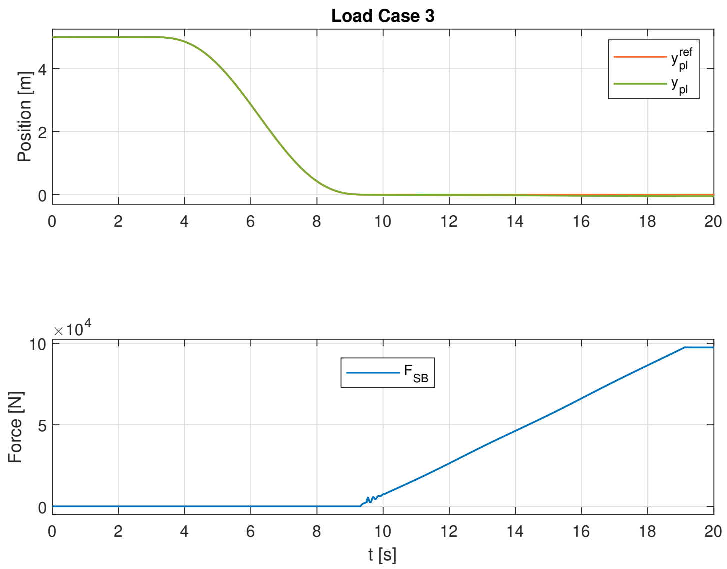


Figure 5.18: Payload position and seabed force during landing i LC3

The third and last load case states that payload should move from 5 m down to, and land on seabed. It was also important to minimize the impact load between payload and seabed with a gently landing. This was done with the same quintic trajectory as for LC2 down to 0 m, then the cascaded (position and velocity) controller is disabled and the constant tension controller takes over. The reference signal for the CT controller is ramped down from 25.5 kN with a slope of -2.5 kN. Which results in the seabed force seen in the lower plot of Figure 5.18, where the upper graph is the payload position.

As stated in 5.4, the CT setpoint is ramped down until desired constant tension and kept there. The tension set during this simulation was the equivalent wire force (at drum) in order to have the weight of the traveling block and sheaves alone, meaning that the weight of the payload rests on seabed. Given that the traveling block with sheaves has a total mass of 600 kg, the equivalent wire force at drum is calculated to be about 1.5 kN.

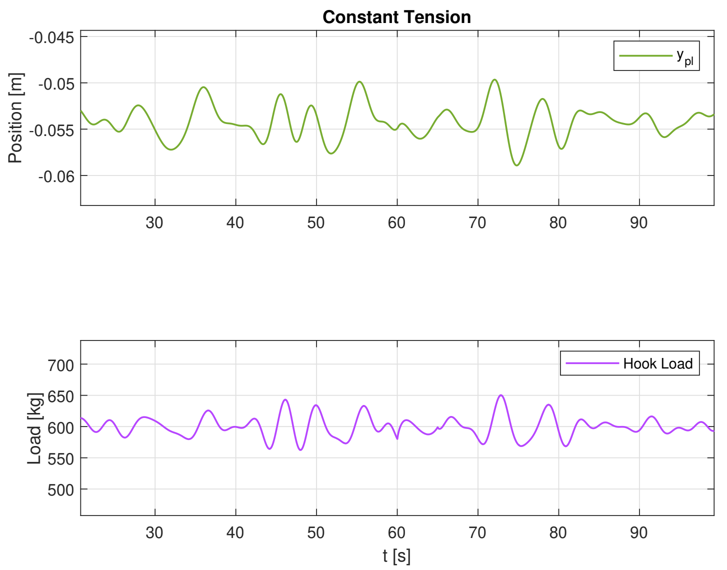


Figure 5.19: Constant tension controller performance

100 seconds of constant tension control as AHC is presented in Figure 5.19, where it is seen that the hook load is kept quite constant around 600 kg, and the payload variation is below 1 cm.

6 Industrial IT

The Simulink-model provides a simplified simulation of the platform, payload and induction motor. In which the payload motion is determined by a control system that receive sensor feedback from the payload and platform, and regulates the motor output by providing a torque reference. This simulation is useful for gaining knowledge about the system behavior. Additionally, designing, implementing and tuning a control system is easier in a simulation due to the fact that there are a wide range of available tools that simplifies the process. Because of this it is common in the industry to develop systems in simulated environments, to later be implemented on the hardware used in the real system while still simulating the plant.

6.1 System Overview

6.1.1 Hardware-in-the-loop Simulation

This technique of simulating the plant while using real hardware is called hardware-in-the-loop simulation (HIL). The hardware that controls the system is usually referred to as the controller. In a HIL simulation the controller communicates with a simulated plant. The communication protocol and interfaces are often equal to the ones being used by the real plant. Which ensures that when the development is near finished, switching from the simulated plant to the real plant should be as seamless as possible.

Figure 6.1 shows a simple model of how the HIL simulation is set up. Path A illustrates the controller communicating with a Real-time target by using the TCP/IP protocol over Ethernet. Path B illustrates the controller communicating with the Plant by using the same communication protocol. An example of a predefined interface is that a motor in the simulation and a motor in the plant use the same reference signal. This interface can be further specified by defining which data type the reference is, and if it is a current, torque or speed reference.

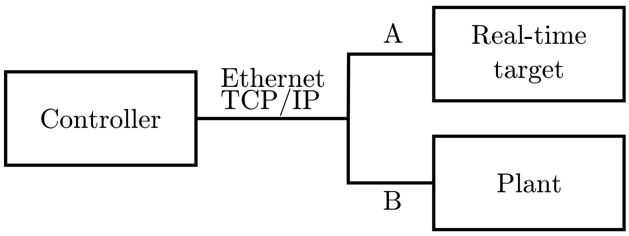


Figure 6.1: Hardware-in-the-loop diagram.

Setting up the HIL simulation is done by removing the control system from the Simulink-model used in 5 and implementing it on the controller. In addition to the control system, a system control logic must be implemented to manage communication with the real-time target, handle user input and operate in different states depending on the functional description of the system.

In order to run the plant (induction motor and drawwork models) on the Real-time target a Simulink model which is able to communicate with the PLC is created. See Appendix XXX. The Simulink model is communicating using TCP/IP. Which mean the IO transferred needs to be mapped equally on both sides of the communication. In order to do that a byte array is defined.

The byte array contains the transferred variables and their respective length. By fixing the length and location of each variable in the byte-array the variable can be read on either side of the communication. Saying the Real-time target outputs two variables of real (float) datatype it will output a byte array with a length of 8 since a real value is 4 bytes long. By knowing the location of the first byte for each value they can be read on the PLC.

Additionally the two machines needs to know what to communicate with. Meaning the IP-addresses needs to match. This configuration is setup in both the PLC and on the Real-time target.

6.1.2 Programmable Logic Controller

In this project the controller used is a Siemens E200S Programmable Logic Controller (PLC). Siemens systems follow the IEC 61161-3 open international standard for PLCs, maintained by the International Electrotechnical Commission (IEC). This standard supports multiple programming languages, which gives the user freedom to implement the logic in several ways. The supported languages are Ladder Diagram (LD), Function Block Diagram (FBD), Structured Text (ST), and Sequential Flow Chart (SFC). Siemens has, however, branded their Structured Text programming language interpretation Structured Contol Language (SCL). IEC 61161-3 does also use an object oriented programming paradigm (OOP), similar to some other programming languages such as Python and C++.

Programmable Logic Controllers are normally used in real-time system, because of their deterministic behavior. They operate using a cyclic manner. Simplified meaning that they read data from input-ports, do their programmed control logic, then outputs data from their output-ports. This is illustrated in figure 6.2. Depending on the amount of data to be read, handled and written, if the PLC has enough power, it ensures that the cycle time will never exceed a given maximum time. This time is dependent on the application. [2]

As mentioned the PLC is to communicate with a Real-time target to perform the HIL simulation. In this project the Real-time target is a Speedgoat. The Speedgoat can run Simulink models in Real-time and therefore communicate with the PLC as described.

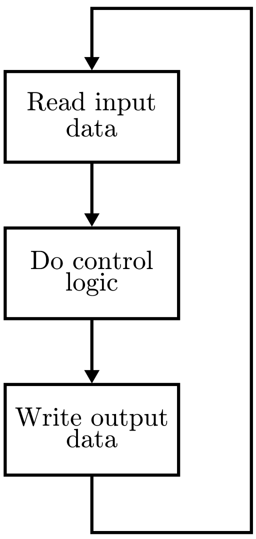


Figure 6.2: PLC cyclic beheavior.

6.1.3 V-model Product Development

According to the V-model of product development the HIL-simulation is used throughout the development from 'Coding, prototyping, and engineering mode' to 'Acceptance tests' [3].

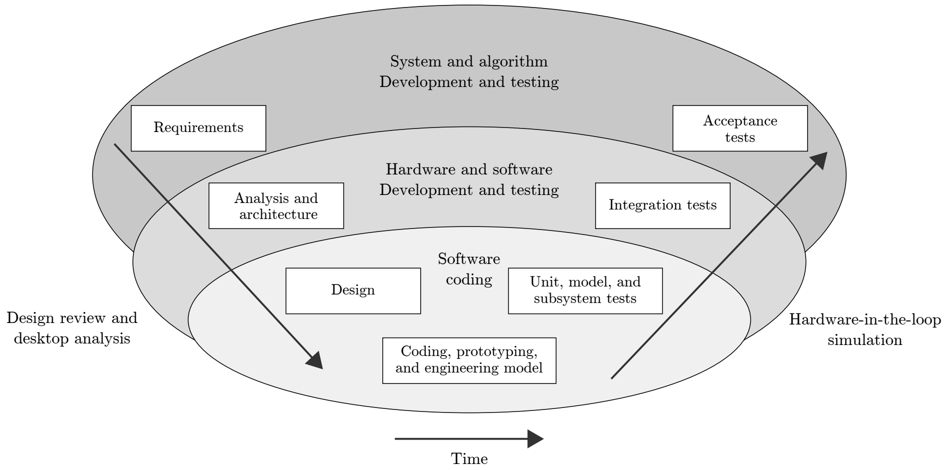


Figure 6.3: V-model for control system design (based on [3]).

The development of the program is done in accordance with the ISO 9001 standard required documentation for an industrial program are Design Specification (DS011), Functional Description (DS021), Program Description (DS024) and Definition of Variables (DS025).

6.2 Design Specification

As mentioned in Chapter 1 Introduction, the project purpose is to virtually model, simulate and control an electrically actuated mechanical system subjected to dynamic loading. How the system should behave is described in the earlier chapters. But in general the design specifications from the project description are: [1].

- The heave compensation must have an accuracy of at least 4 cm.
- The PLC must communicate with Real-Time Speedgoat by using TCP/IP.
- The Speedgoat must run the Simulink simulation in real-time.
- The system must include an operator HMI.
- The system must include a management HMI.

Additional system requirements to further further improve the system:

- The system must have an emergency button that shuts down all current operations.
- The system must be able to apply a brake on the motor, which fully stops the motor movement.
- The user must be able to turn the system on and off. When the system is off, the brake must be active.
- The user must be able to activate and deactivate heave compensation.
- The user must be able to manually jog the payload.
- The user must be able to automatically move the payload using a custom trajectory. Heave compensation must be activated during this mode.

In order to operate the system an HMI must be created. The following list contains the user interface requirements:

- The HMI must be creatied using WinCC RT Advanced.
- The HMI must be easy to understand.
- The HMI must display some Key Performance Indicators.
- The HMI must contain a service setting to change controller parameters.
- The service setting must be password protected.
- The HMI must be able to switch between manual and auto mode.
- A management HMI using PI ProcessBook must be created.
- The management HMI must be easy to understand.
- The management HMI must display some Ket Performance Indicators.

6.3 Functional Description

Based on the system requirements a logical design approach is to implement a state machine. A state machine does different logic depending on which state is active. The active state may transition to another state if a condition or a set of conditions are met. The system is also going to serve a Human Machine Interface (HMI), run a control system and send/receive data independent on the state machine. Therefore the state machine must be ran in a sequence together with the other mentioned logic.

6.3.1 System Flow

The main system flow is time-independent and runs continuously. The cyclic system flow is time dependent and is running every 10 ms, this is because some logic requires a fixed cycle-time. These system flows are represented as flowcharts in Figure 6.4 and Figure 6.5, respectively.

The main flow first handles the communication with the real-time target. This involves receiving and sending data. Next, the HMI is updated with the new information from the real-time target, as well as reading the input from the user. Some of the real-time target data may need to be processed before it can be used. Examples of this are data that needs to be filtered, integrated, or differentiated. This is done in the Data processing block. Lastly, the State machine is executed, acting on the data from the real-time target, user input, and processed data.

The cyclic system flow starts by setting a send flag true. This flag is related to the Communication block in the main system flow. Whenever this flag is set to true, the Communication block sends data and sets the flag to false. Meaning that this system sends data every 10 ms. Furthermore, the Control system is executed. The Control system requires a fixed step-time to work properly.

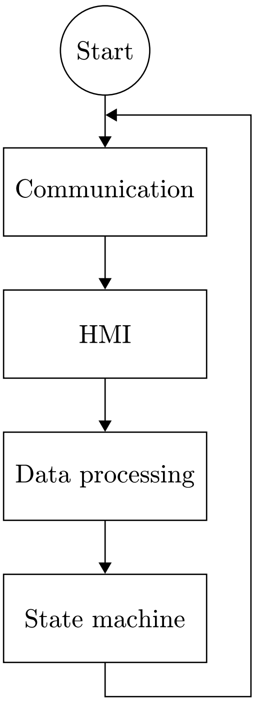


Figure 6.4: Main system flow.

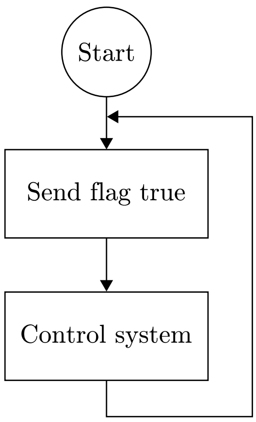


Figure 6.5: Cyclic system flow.

6.3.2 State Machine

The state machine has five states: *off*, *idle*, *manual*, *auto* and *service*. These states and how they connect are illustrated in Figure 6.6. Initially the active state is *off*. TCx denotes the transition conditions.

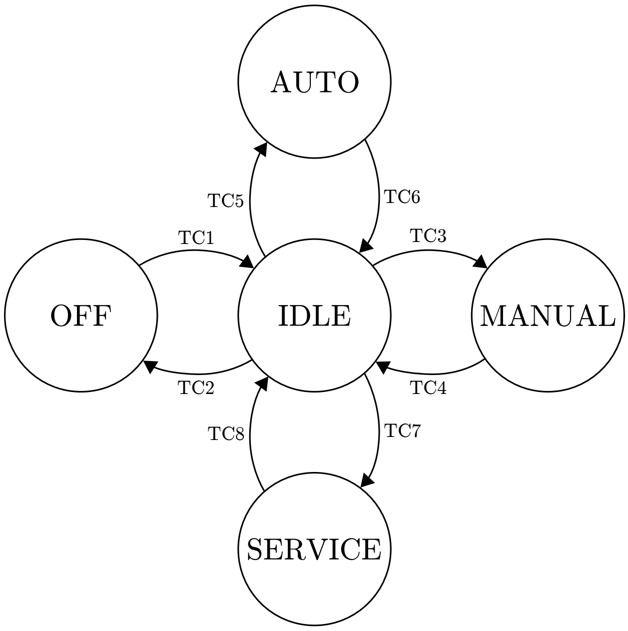


Figure 6.6: State transition diagram.

6.3.3 States

What happens in the different states is important to understand the logic of the system.

Off
In the *off* state, there are no ongoing operations and the brake is always active.

Idle
In the *idle* state, the brake is deactivated and the control system is initially trying to keep the motor at rest, meaning that the payload will follow the platform movement. This state is to be considered an intermediate state, where the state machine awaits user input for further operation. The user is able to activate or deactivate heave compensation.

In addition, the user can prepare a trajectory for the *auto* state by entering a desired setpoint and a time to reach that setpoint. The setpoint is limited to minimum 1 m and maximum 14 m relative to the seabed. The minimum time to reach the setpoint is linearly dependent on the distance from the current position to the setpoint. The system then generates a trajectory by interpolating between the points using a fifth-order polynomial. The trajectory is only generated and not set in motion until the user choose to run the trajectory.

Manual
In the *manual* state, the payload position and velocity can be changed manually. The payload position is always limited to minimum 1 m and maximum 14 m relative to the seabed. To change the position, heave compensation must be active. The position change is relative to seabed. To change the velocity, usually referred to as jogging, heave compensation may either be active or inactive. If heave compensation is active the payload velocity can be changed to ± 1 m/s relative to the seabed. If heave compensation is inactive the payload velocity can be changed to ± 1 m/s relative to the platform.

Auto
The *auto* state is only active while running a trajectory. Heave compensation is always active in

this state.

Service
In the *service* state, the system condition may be monitored. In addition the control system parameters gains may be altered. In this state heave compensation is always active.

6.3.4 Transitions

A transition that is not represented in the diagram is what happens when the emergency button is pressed. As stated in the requirements all current operations must shut down when this happens. As the emergency button may be pressed in every state, there exist a transition from every state to the *off* state. In the *off* state the brake is turned on, and all ongoing logic shut down. The state machine is unable to change out of the *off* state when the emergency button is pressed. Which means that the emergency button must be pressed again to unlock normal behavior.

When starting up the system the active state is *off*, as previously mentioned. The only transition possible is to the *idle* state. The state machine transitions from *off* to *idle* when the on-button is pressed (TC1). After the transition the active state is *idle*, if the user press the on-button in this state, the state changes back to *off* (TC2).

To change from *idle* to *manual* (TC3), the manual-button must be pressed. If the active state is *manual* and the manual-button is pressed the state changes to *idle* (TC4).

To change state from *idle* to *auto* (TC5), heave compensation must be active, a trajectory must be generated and the run-button must be pressed. The state machine will automatically switch back to the *idle* state when the trajectory is finished or if the cancel-button is pressed (TC6).

To change state from *idle* to *service* (TC7), heave compensation must be active, the user must be logged in and the service-button must be toggled on. When the *service* state is active and the service-button is toggled off, the state changes to *idle* (TC8).

6.4 Program Description

The program description describes how the functionalities in the previous section are implemented in the relevant system. Since the controller the logic is being implemented on a Siemens PLC, the program is written by using the Totally Integrated Automation Portal (TIA). The TIA Portal is Siemens proprietary software for developing software for their PLCs. Regarding programming languages, the program is written using a combination of Functional Block Diagram and Structured Control Language (SCL). Almost all logic is written using SCL, while FBD has been used to tie everything together. This is because the FBD programming language is a graphical programming language, and can quickly become convoluted.

6.4.1 Organization Blocks

Organizational blocks (OB) are the interfaces between the operating system and the user program, According to Siemens. Siemens has a set of predefined organizational blocks that each serves their own purpose. For example, OB1 is the default block, which is run each cycle, and OB100 is a block that only runs when the operating mode of the CPU is changed from STOP to RUN. In this system, three organizational blocks are used: OB1, OB35, and OB100. OB35 is a cyclic interrupt block, which means that this block runs consistently at defined intervals. OB35 is set to run every $10ms$. As written in Chapter 6.3.1, the main flow of the system runs continuously, therefore these blocks are placed in OB1. The send data flag and Control system on the other hand is time-dependent, and are placed in OB35.

6.4.2 Data Blocks

Data blocks (DB) are not in accordance with the IEC 61131-3 standard, but rather Siemens specific. DBs are used to store data in memory, which makes it accessible to all Organizational Blocks, Function Blocks, and Functions. An alternative to data blocks are tags, data saved in tags is also available globally. This data on the other hand not saved in memory, but in M-memory. Saving data in M-memory has some disadvantages, an example being that there are no security against overlapping variables. Additionally, the m-memory storage space is limited; consequently, a common practice is to prefer data blocks over tags. [2]

6.4.3 Function Blocks

Function blocks (FB) are blocks of code with internal memory that may be instantiated, much like a class in other OOP languages. For each instance of the FB, a data block is created, which stores all static variables. A naming convention when creating function blocks is to lead with the function block abbreviation (FB), with a following underscore (FB_), and at last the function block name in camel case form (FB_SomeFunctionBlock).

CONT_C (FB)

The control system is implemented by using the built-in CONT_C function block, Siemens' discrete ideal PID-controller. Since the control system is a closed loop cascaded controller, two CONT_C function blocks are used in series with the output of the position regulator being the reference of the velocity regulator. The blocks has multiple features, such as integration hold, output saturation, and the possibility to enable and disable. Figure 6.7 shows how the blocks are used in the organization block using functional block diagram. Not shown in the figure, is the integral clamping. Clamping the integral is done to avoid accumulating a large integral contribution. This is implemented by checking whether the integral contribution is greater or less than than the max and min saturation, respectively. When the integration term exceeds these limits, the integration hold flag is set high, such that the controller stops integrating. When the integral contribution falls within the max and min limit, the integration hold flag is set low, and the integration proceeds as usual.

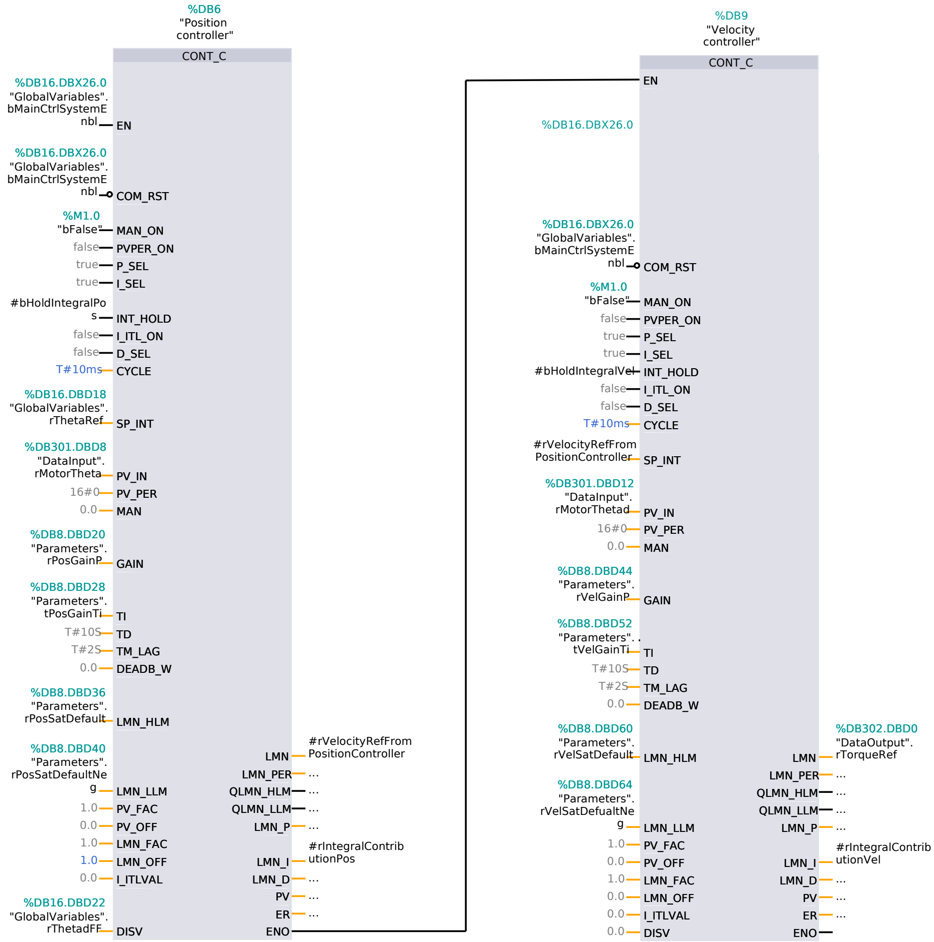


Figure 6.7: Control system implemented with built-in blocks.

Communication (FB)

The first block in the main system flow, the Communication block, is implemented as an FB written using FBD. The Communication block is not custom, but is obtained from a library. Accordingly, this block will not be explained in great depth. The purpose of this block however, is to establish TCP/IP communication with the real-time target. And handle incoming and outgoing data.

Figure 6.8 show how the TCP endpoint is configured by defining the target IP address and port number. Figure 6.9 and 6.10 show how the send and receive functions are configured, respectively. The *LEN* and *DATA* inputs refer to the length of the data and where it is read and written from, respectively.

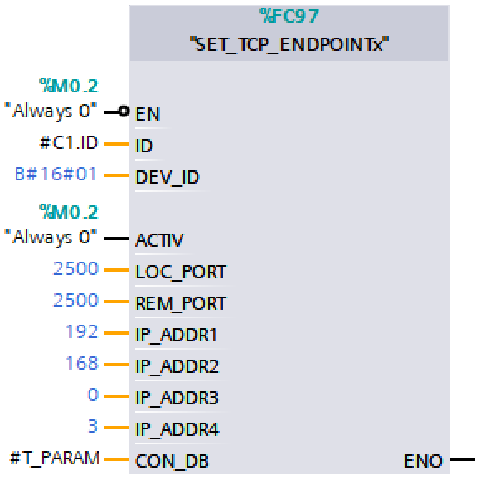


Figure 6.8: Set TCP endpoint.

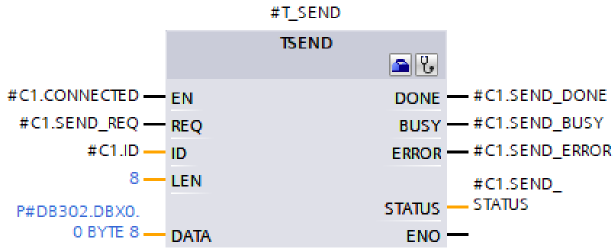


Figure 6.9: Send function.

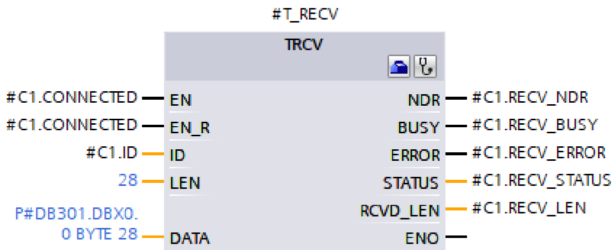


Figure 6.10: Receive function.

Operator HMI (FB)

The second block in the main system flow is the HMI block. This block is implemented as a FB and written using SCL, and has no inputs or outputs. The HMI FB stores all variables used in the HMI. Then if something changes elsewhere in the code, the HMI block is the only code that needs to be updated. Since this block is also the users way of interacting with the system, there is implemented logic which updates the system state depending on the user interaction. To avoid problems where the system state and user input do not agree of what to do at certain times, the system is given priority. The logic flow of this implementation is visualized in Figure 6.11.

First, data to be presented to the user is read from around the system. If the data requires conversions or processing to be represented in a more readable manner, that is done at this stage. Next, the current system state is checked against the requested system state. If these do not match, the system has requested a state change and is given priority to go through with that change. In other words, the HMI FB updates its current state to match the system. If the system state and required state is the same, then the HMI FB updates the system with the latest user input. This is done by directly setting variables in the state machine FB. At last, the HMI FB updates the buttons to be enabled or disabled depending on what state the system is in. An example is that in the auto-state, the user should not be able to disable heave compensation.

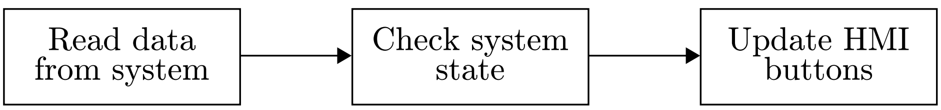


Figure 6.11: Logic flow of HMI function block.

Data Processing (FB)

The third block in the main system flow is the Data processing block. It is implemented as a FB and written using SCL. This block takes two inputs, the platform acceleration and the main cycle last cycle time, as shown in Figure 6.12. With this data it calculates the platform velocity by performing numerically integration using the Forward Euler algorithm. After the velocity is found, the signal is filtered, using a low-pass filter. The current platform velocity can then be retrieved from anywhere in the code by accessing this block.

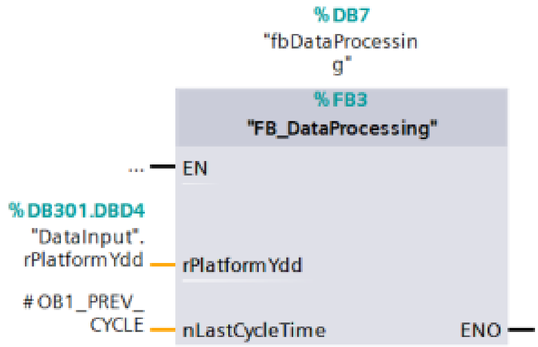


Figure 6.12: Data processing function block.

State Machine (FB)

The fourth and last block in the main system flow is the State machine block. This block implements the state machine from Chapter 6.3.2 as a FB, and is written using SCL. It takes a single input, the previous cycle time, and outputs the motor angle reference, angular velocity feed-forward, and a reset controller flag, as shown in Figure 6.6.

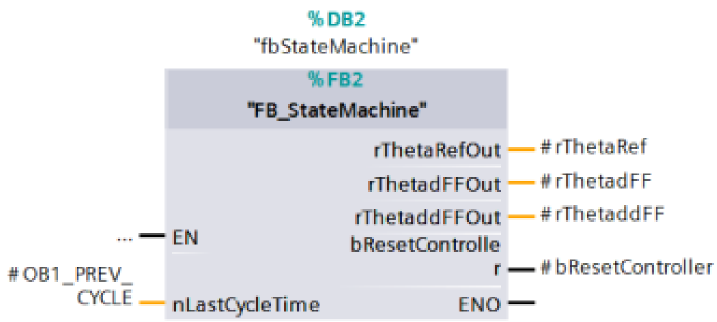


Figure 6.13: State machine function block.

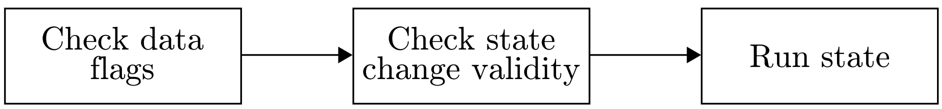


Figure 6.14: Logic flow of State machine function block.

The State machine logic flow is illustrated in Figure 6.14, divided into three steps. In the first step, the data flags are checked and compared with the current state by using an IF-statement. One example of these data flags is the system on flag. The system on flag is true if the system is on, and false if the system is off. Supposed that the current state is *off* and the system on flag is false, then nothing should happen. If the current state is *off* and the system on flag is true, then the system should turn on. The logic will then set a request state variable to request the state *idle*. The same logic can be implemented in reverse to turn the system off. If the current state is *idle* and the system on flag is false, then the requested state should be *off*. This example is shown in Listing 1, and is a part of how the first block in the State machine is implemented.

Listing 1: IF-statement example in Structured Control Language.

```
IF (nCurrentState = #OFF) AND bSystemOn THEN
    nRequestState := #IDLE;
ELSIF (nCurrentState = #IDLE) AND NOT bSystemOn THEN
    nRequestState := #OFF;
```

```
END_IF;
```

The main purpose of the second step is to check whether the system is allowed to go from the current state to the requested state. If not, the requested state is set equal to the current state. This step is only executed if the current state is not equal to the requested state, as that requires to check the state change validity. This logic is implemented by using a combination of IF-statements and CASE-statements. Continuing with the example from the previous section, with the current state being *off* and the requested state being *idle*, shown in Listing 2. Since the current state and requested state are different, the CASE-statement is executed. The CASE-statement checks the current state, and then the IF-statement checks if the requested state is allowed in the current state. Requesting *idle* is allowed, and the state is changed to *idle*.

Listing 2: CASE-statement example in Structured Control Language.

```
IF nCurrentState <> nRequestedState THEN
    CASE nCurrentState OF
        #OFF:
            IF nRequestedSate = #IDLE THEN
                nCurrentState := #IDLE;
            END_IF;
        #IDLE:
            IF nRequestedState = #OFF THEN
                nCurrentState := #OFF;
            END_IF;
        ELSE
            nRequestedState := nCurrentState;
    END_CASE;
END_IF;
```

The third and last step runs the current state. This step is implemented with a CASE-statement and implements all logic that is happening in each state. Listing 3 shows how...

Listing 3: Run current state CASE-statement.

```
CASE nCurrentState OF
    #OFF:
        // this code executes in the off-state
    #IDLE:
        // this code executes in the idle-state
    #MANUAL:
        // this code executes in the manual-state
    #AUTO :
        // this code executes in the auto-state
    #SERVICE:
        // this code executes in the service-state
END_CASE;
```

Quintic (FB)

The trajectory executed in the *auto*-state is implemented as a fifth-order polynomial interpolated between two points. The derivation of this polynomial is done in Chapter 3.2.1. Since computing linear algebra on the PLC is not natively supported, the expressions are generated symbolically in MATLAB and implemented as a function block. Since each trajectory starts and ends with zero velocity and acceleration, the symbolic expressions for position and velocity are reduced. The implemented expressions are shown in Listing 4. The variables are summarized in Table 6.2. The current time is incremented by the system last cycle time each cycle, and is reset every time the quintic trajectory is completed or canceled. The trajectory time, initial position, and end position are set every time a trajectory is generated in the *idle*-state, such that the quintic FB is ready whenever the user presses the run-button.

Listing 4: Quintic function block position and velocity expressions.

```
#rYRef := #rY0 - (10 * #rT ** 3 * (#rY0 - #rY1)) / #rTp ** 3 + (15 * #rT ...
    ** 4 * (#rY0 - #rY1)) / #rTp ** 4 - (6 * #rT ** 5 * (#rY0 - #rY1)) / ...
    #rTp ** 5;
#rYdRef := (60 * #rT ** 3 * (#rY0 - #rY1)) / #rTp ** 4 - (30 * #rT ** 2 * ...
    (#rY0 - #rY1)) / #rTp ** 3 - (30 * #rT ** 4 * (#rY0 - #rY1)) / #rTp ** 5;
```

Table 6.1: Variables in quintic expressions implemented on the PLC.

Variable:		Description:	Unit:
$rYRef$	-	Position reference	m
$rYdRef$	-	Velocity reference	$\frac{m}{s}$
$rY0$	-	Initial position	m
$rY1$	-	End position	m
rTp	-	Trajectory time, from initial to end position	s
rT	-	Current time, starting from zero	s

Ramp (FB)

In the *manual*-state, the user is able to jog the payload. To avoid sudden changes in velocity, the velocity reference is given by a ramp from zero to a given end velocity, in positive or negative direction. The velocity reference is ramped back to zero, when the user let go of the jog-buttons. This ramp functionality is implemented as a function block, written using SCL. Listing 5 show the ramp function block code. First an IF-statement is used to check whether the end-value has changed, if it has, the slope is updated. If the output value is within a tolerance of the end-value, the output reference is set equal to the end-value. If not the output reference is incremented by integrating the slope with the last cycle time.

Listing 5: Function block that generates a ramp reference.

```
// Check if end value has changed
IF #rEndValueMem <> #rEndValue THEN
    #rRamp := (#rEndValue - #rCurrentValue) / #rRampTime;
    #rEndValueMem := #rEndValue;
END_IF;

// Ramp
IF ABS(#rEndValue - #rCurrentValue) > #rTolerance THEN
    #rReference := #rReference + #rRamp * #rLastCycleTime;
ELSE
    #rReference := #rEndValue;
END_IF;
```

Table 6.2: Variables in quintic expressions implemented on the PLC.

Variable:		Description:	Unit:
$rReference$	-	Output reference	m
$rEndValue$	-	End-value	$\frac{m}{s}$
$rRampTime$	-	Ramp time	m
$rSlope$	-	First order slope coefficient	m
$rLastCycleTime$	-	Last cycle time	s
$rTolerance$	-	Tolerance to end ramp	s

Low Pass Filter (FB)

A noisy signal can often cause disturbances in the system. A solution to deal with high frequency noise, is to use a low-pass filter. This filter allows signals with low frequency to pass, and blocks signals with high frequency. The cost if this filtering technique is that the output signal is delayed, depending on how much of the signal is filtered. Therefore, when tuning the cutoff frequency, an intermediate value that removes most of the noise while minimizing the delay is desirable.

In the frequency domain a low-pass filter is implemented as the following transfer-function.

$$H(s) = \frac{Y}{X} = \frac{\omega_c}{s + \omega_c} \tag{6.1}$$

By multiplying with the denominator on both sides, and using inverse Laplace, the transfer-function is converted to time domain.

$$\frac{dy_i}{dt} + \omega_c y_i = \omega_c x_i \tag{6.2}$$

The derivative expression is converted to its discrete form using backwards finite difference derivative approximation. Where dt is the stime-step.

$$\frac{dy_i}{dt} \approx \frac{\nabla y}{dt} = \frac{y_i - y_{i-1}}{dt} \tag{6.3}$$

The differentiation is expanded, and the equation is rewritten. Leading to an expression of the next filter output.

$$\frac{y_i - y_{i-1}}{\omega_c dt} = x - y_i \tag{6.4}$$

$$y_i = x_i \left(\frac{\omega_c dt}{1 + \omega_c dt} \right) + y_{i-1} \left(\frac{1}{1 + \omega_c dt} \right) \tag{6.5}$$

This equation is simplified by defining the coefficient α.

$$y_i = \alpha x_i + (1 - \alpha) y_{i-1}, \quad \alpha = \frac{\omega_c dt}{1 + \omega_c dt}. \tag{6.6}$$

The low-pass filter is implemented as a function block, since the equation requires to store the last output value. The implementation is shown in Listing 6. The function block has three inputs, the cutoff frequency, time-step, and current measured value. On the first line, the cutoff frequency is converted from hertz to radians per second. Next, the α coefficient is calculated, before the output is calculated according to Equation 6.6. At last, the current output is stored to be used on the next cycle.

Listing 6: Low-pass filter implemented as a function block.

```
// Calculate omega
#rOmega := 2 * #rPi * #rCutoffFrequency;

// Calculate dynamic constant
#rAlpha := (#rOmega*#rDt)/(1 + #rOmega*#rDt);

// Calculate output
#rY := #rAlpha*#rX + (1 - #rAlpha)*#rYLast;

// Save output as last output
#rYLast := #rY;
```

Haugen has suggested that the time-step should be considerably smaller than the filter time-constant. The time-step is then selected to be in accordance with the following expression. [5]

$$dt \leq \frac{\tau}{5}, \quad \tau = \frac{1}{\omega_c} \tag{6.7}$$

6.4.4 Functions

Functions (FC) are blocks of code without internal memory. This means that a function does not require a data block. The naming convention for function is prefix the function name with a capital letter F, followed by an underscore. The function name should be in camel case (F_SomeFunction), similar to the naming convention for function blocks.

Change State (FC)

Listing 1 gave an example on how the states are requested based on the current state and the data flags. The request state where then directly set by changing the variable. However, sometimes during a state change it is required that some logic happens, just once. This is implemented with a function, that takes the requested state as an input. This implementation is shown in Listing 7.

Listing 7: Change state function.

```
// Function call
"F_ChangeState"(nRequestState := #OFF); // request state off

// Function definition
"fbStateMachine".nRequestState := nRequestState;

CASE nRequestState OF
    #OFF:
        // this code executes on change to off-state
    #IDLE:
        // this code executes on change to the idle-state
    #MANUAL:
        // this code executes on change to the manual-state
    #AUTO :
        // this code executes on change to the auto-state
    #SERVICE:
        // this code executes on change to the service-state
END_CASE;
```

Sign (FC)

A useful function not included in the Siemens library, is the sign-function. The sign-function takes a single input and returns the sign of that input. Which implies that the function returns 1 if the input is positive, and -1 if the input is negative. This is implemented as a function, as there are not need to store any data between each use. Listing 8 shows the implementation of the sign-function as a function.

Listing 8: Sign-function in SCL.

```
#F_Sign := #rValue / ABS(#rValue);
```

Quintic Max Velocity (FC)

While the user is generating trajectories, it may be useful to know what the maximum velocity of the current trajectory is. Since the quintic trajectory is predefined to begin and end at zero velocity and acceleration, the maximum velocity is going to be in the halfway-point of the trajectory-time. As a result, the maximum velocity is calculated by using the velocity expression from the Quintic function block implementation, and inserting the current trajectory-time divided by two. This implementation is shown in Listing 9.

Listing 9: Quintic maximum velocity.

```
#rT := #rTp / 2.0;
```

```
#F_QuinticMaxVel := (60 * #rT ** 3 * (#rY0 - #rY1)) / #rTp ** 4 - (30 * ...
    #rT ** 2 * (#rY0 - #rY1)) / #rTp ** 3 - (30 * #rT ** 4 * (#rY0 - ...
    #rY1)) / #rTp ** 5;
```

Quintic Max Acceleration (FC)

As well as the maximum velocity, knowing the maximum acceleration of the generated trajectory is also useful. The maximum acceleration is when the jerk of the trajectory is zero. This expression is shown in Equation 6.8.

$$\frac{d^3y}{dt^3} = 6x_4 + 24x_5t + 60x_6t^2 = 0 \tag{6.8}$$

This equation is solved symbolically in MATLAB, resulting in the following expression for the time where the trajectory has maximum acceleration.

$$t_{a,max} = \frac{t_p(\sqrt{3}+3)}{6} \tag{6.9}$$

The implementation is shown in Listing 10.

Listing 10: Quintic maximum acceleration.

```
#rTa := (#rTp * (SQRT(3) - 3) / (6));
#F_QuinticMaxAcc := (180 * #rTa ** 2 * (#rY0 - #rY1)) / #rTp ** 4 - (60 * ...
    #rTa * (#rY0 - #rY1)) / #rTp ** 3 - (120 * #rTa ** 3 * (#rY0 - #rY1)) ...
    / #rTp ** 5;
```

6.5 Definition of Variables

Variables can be defined in several ways on a Siemens PLC, depending on the usage. As mentioned in Section 6.4.2, there is a common practice to prefer data blocks over tags. Inside a single data block, the only available variable type is static variable. Static variables are remembered across cycles, which means that variables in a data block is never changed if not done implicitly. Function blocks may also have static variables, as each instance of a function block has a data block. An example usage of when to use a static variable is in the low-pass filter function block implemented on page 54. In the low-pass filter function block, the previous output value is stored to be used in the next cycle.

Another variable type is temp, short for temporary. These are variables which data is forgotten or deleted each cycle. Temporary variables can be defined in functions, function blocks and organization blocks. Data blocks however, can not store any temporary variables. Temporary variables are used when after the execution the value of the variable does not matter, and that the variables is calculated again next cycle. Examples of using temporary variables are when doing intermediate calculations. In organizational blocks, function blocks and functions it is possible to create constant variables. A constant variable is a variable whose value never changes. Trying to write to a constant variable will only result in error messages. These variables are typically used for data that is predefined, and is known to never change during the operation of the system. Examples of such variables are the dimensions of a physical part of the system.

All variables created in the system uses a common naming convention, summarized by the following table.

Data type:		Prefix:	Example:
Boolean	-	b	bMyBool
Int	-	n	nMyInt
Real	-	r	rMyReal
Time	-	t	tMyTime
String	-	s	sMyString

Four data blocks are created to store variables used across the system: data input, data output, parameters and global variables. The data input DB stores all incoming data. This includes payload position, platform acceleration, motor power usage, amongst others. The data output DB stores all outgoing data. Two variables are defined in this data block: torque reference for the motor and brake disable. If the brake disable is true, the brake is not physically in contact with the system. The parameters DB contains all system parameters, a few examples being gear ratio, default position controller gain, and drum radius. The global variables DB contains all system variables that often changes and are used frequently. Examples of global variables are the send data flag, energy usage, system timeout, system uptime. All implemented variables can be seen in Appendix F.

6.6 Human Machine Interface

6.6.1 Operator HMI

The operator HMI is the operators way of manipulating the system according to the functional description. To improve the usability the HMI is divided into four pages: home, manual, auto and service. Each page serves their purpose, but are designed using the same template.. This results in some parts of the pages containing the same information and buttons. On the top of the template page, there is a navigational section to navigate between the different pages. The left side of the template contains the on/off-button, the active heave compensation enable switch, and a soft emergency button. In addition, there are indicators to let the operator know which state is the active state. On the bottom of the template there is a plot of the payload. Some of the pages includes some additional data in this plot.

Home

The home page serves as a purely informational page. There is an animation of the payload movement, together with a position setpoint and the current error on that setpoint in centimeters. Along with this there are several indicators of the payload and platform motion, motor power output and speed, and wire force acting on the drum. Figure 6.15, 6.15 and 6.17 show how the HMI looks while in off, idle without and with active heave compensation.

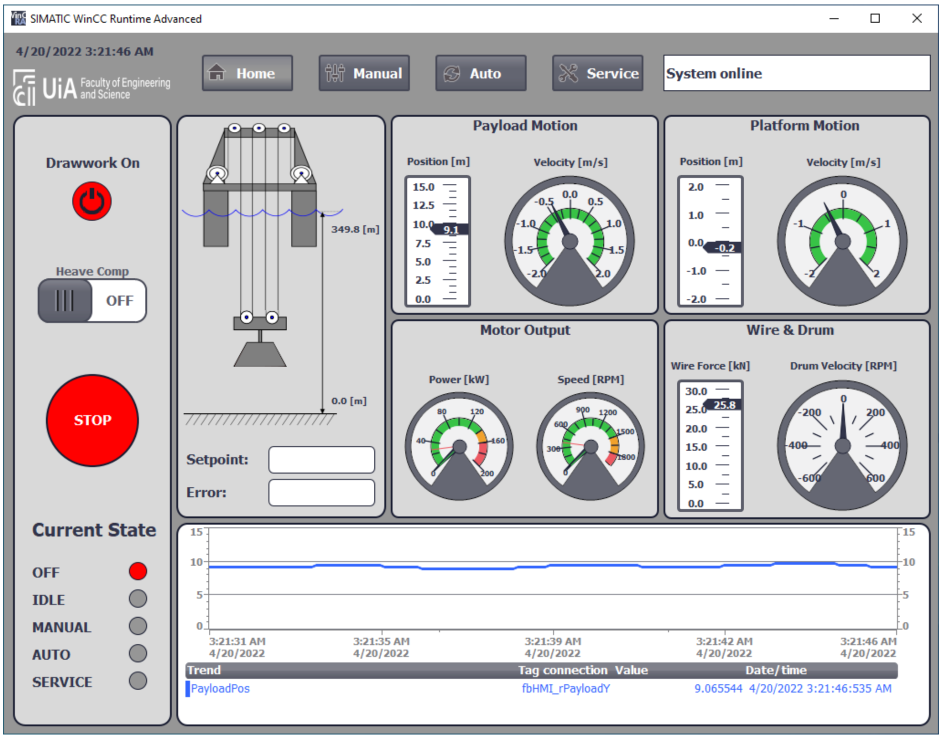


Figure 6.15: Operator HMI home page, system off.

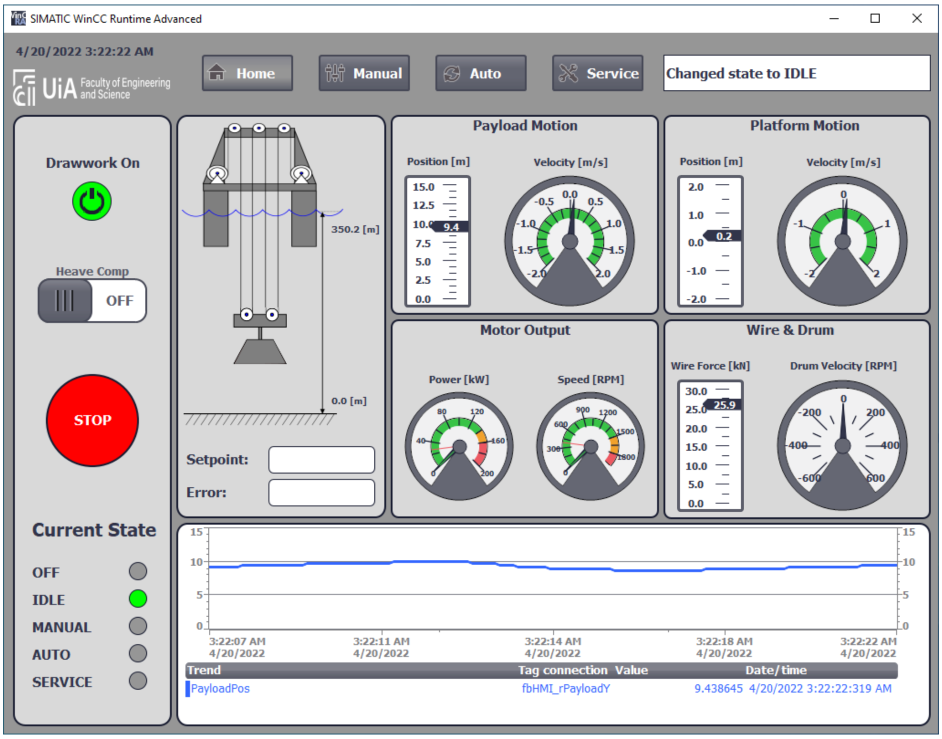


Figure 6.16: Operator HMI home page, system on.

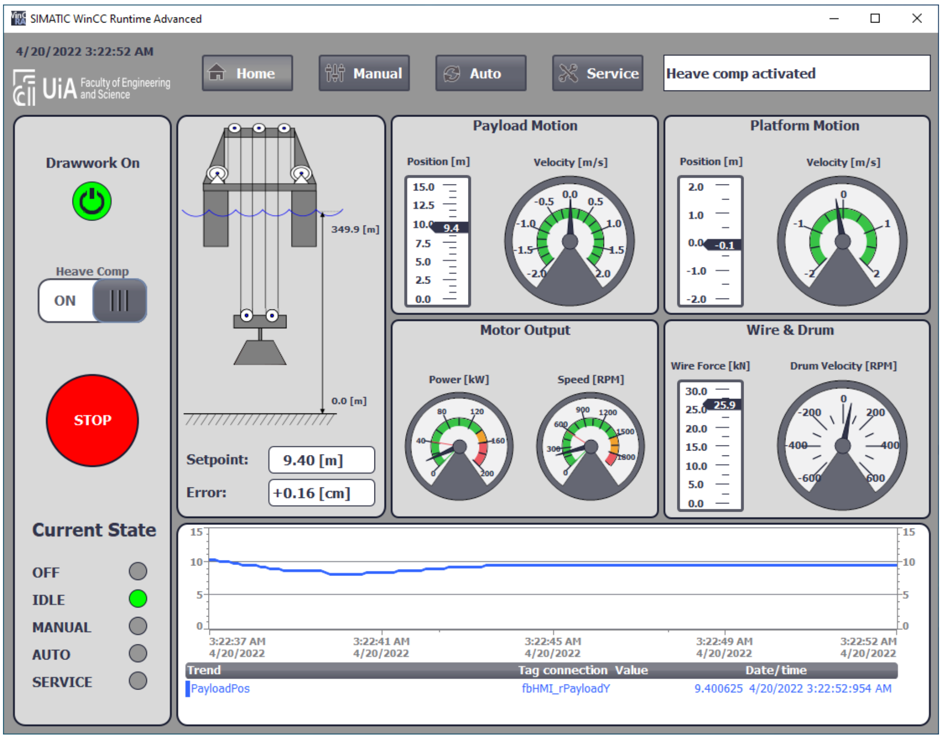


Figure 6.17: Operator HMI home page, active heave compensation enabled.

Manual

The manual page represents the control available in the system *manual* state. As with the home-page, the manual page has an animation of the payload movement. To enable the manual system state, the operator must press the manual-button next to the text *Manual Payload Control* label. In the manual page the operator is able to jog the payload up and down by using the plus and minus buttons, both with active heave compensation activated or deactivated. To move the payload by using position setpoints, active heave compensation must be enabled. As seen in Figure 6.18 and 6.19. Whenever the user exits the manual page, the system state automatically switches to *idle*.

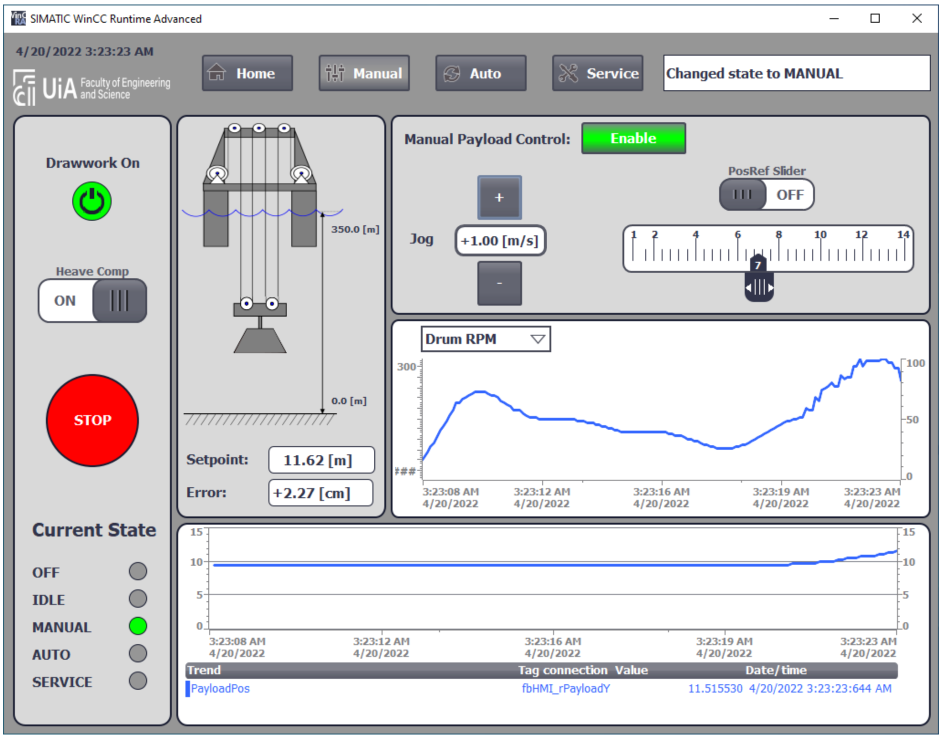


Figure 6.18: Operator HMI manual page, velocity jogging.

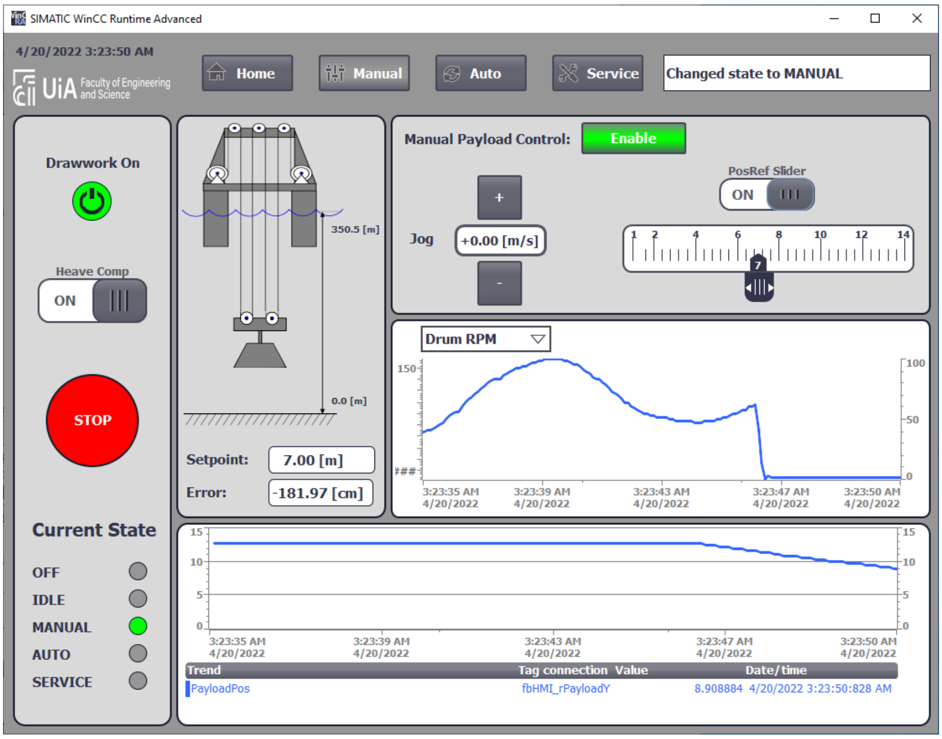


Figure 6.19: Operator HMI manual page, position reference.

Auto

On the auto-page the operator can generate a custom trajectory from the current position to a position between 1m above seabed to 14m above sea bed. This limit is a soft internal limit. How the trajectory is generated is discussed earlier in this chapter. To generate a trajectory, the end position and trajectory time is set by using the plus and minus buttons next to the text field. Pressing the buttons marked as predefined fills the text field with a predefined internal values.

When the operator is ready, the trajectory is generated by pressing the green button with the *Generate* label. This button turns green when a trajectory can be created from the given data in the text field.

After generating a trajectory, an illustration on the right hand side conveniently show the direction of the trajectory, as well as the maximum and minimum accelerations, and the maximum velocity. To run the generated trajectory, the operator can press the green *Run*-button. The *Run*-button is greyed out and can not be pressed if a trajectory is not generated. This is shown in Figure 6.20. While running a trajectory the operator has the option to cancel at any given time by pressing the red *Cancel*-button. This stops the trajectory and automatically switches the system state to *idle*. This is shown in Figure 6.21.

One of the predefined buttons is marked with *Land*. When this button is activated, it turns green, and a landing sequence is initiated. This sets the end position below the previously defined soft limit, at zero meters, or the seabed. If the system is set to run in while the *Land*-button is active, the *auto*-state does not stop when reaching the seabed, but rather initiates the next step of the landing procedure, lowering the payload using a constant tension controller to unload the payload. In addition to the payload position, the operator is able to inspect the estimated seabed force acting on the payload, as well as the hook load, acting on the wire. Figure 6.22 shows how the auto-page looks after a landing is completed.

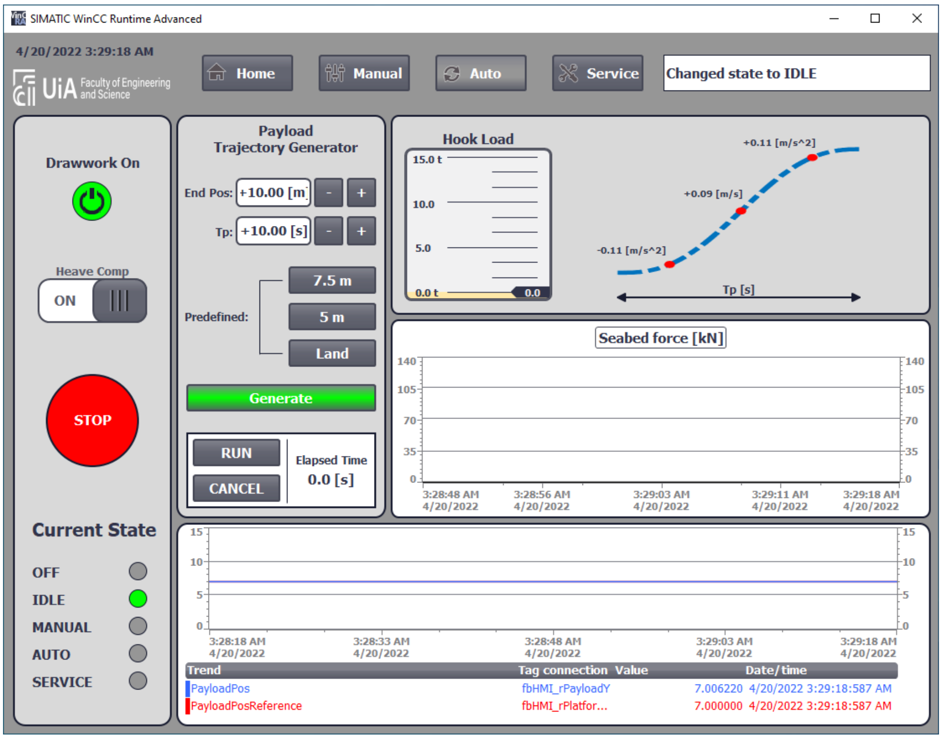


Figure 6.20: Operator HMI auto page, generating trajectory.

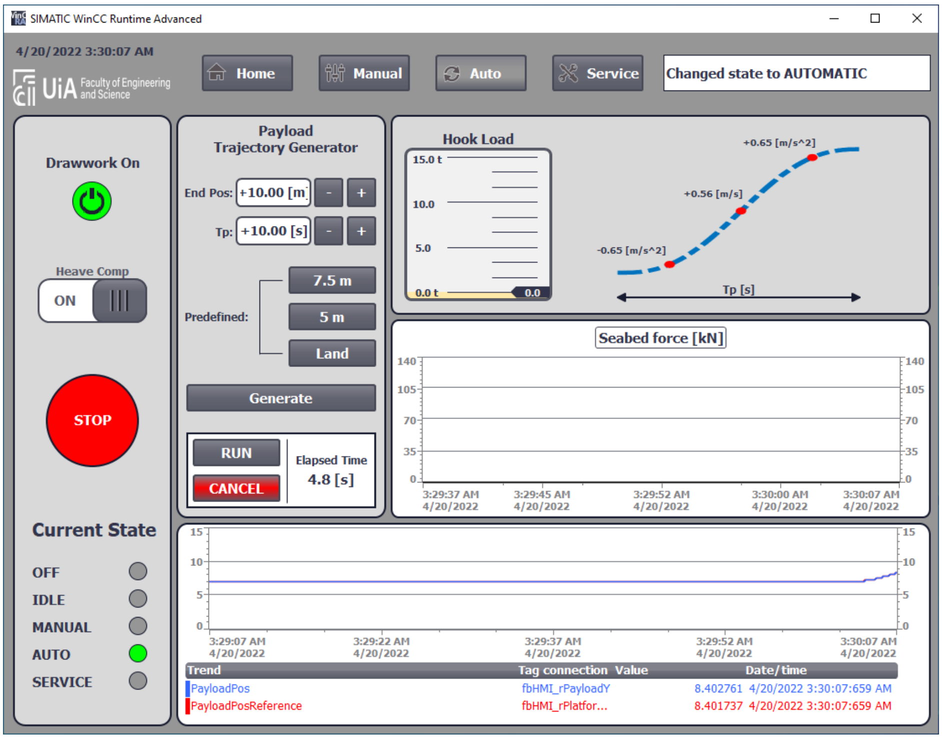


Figure 6.21: Operator HMI auto page, running trajectory.

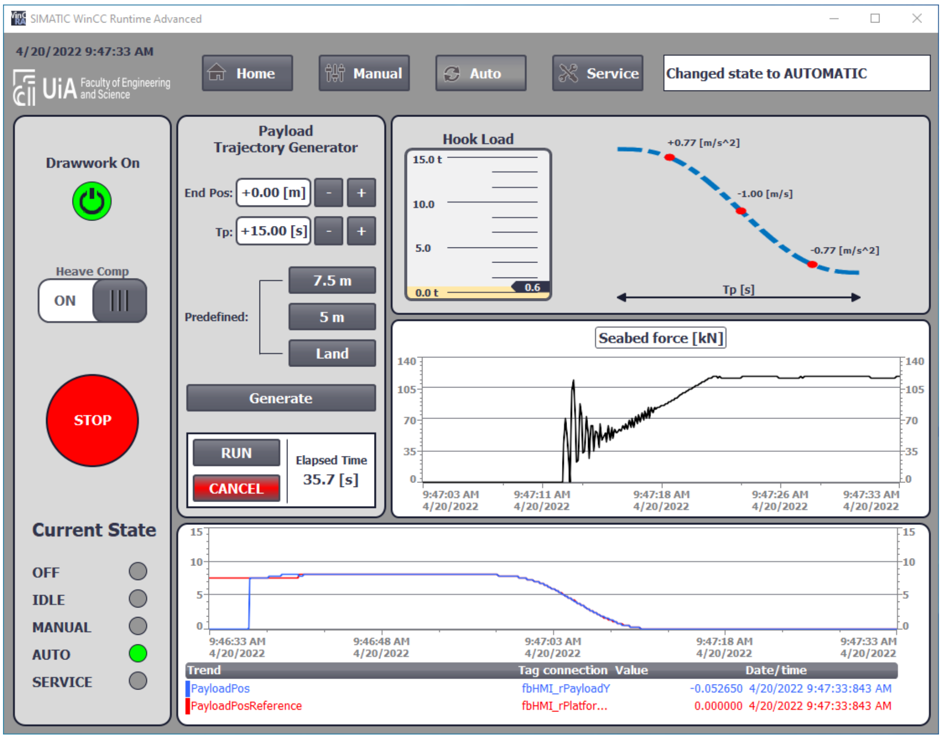


Figure 6.22: Operator HMI auto page, landing payload.

Service

The service page is password protected to ensure that no parameters are changed by accident. On this page the operator is able to adjust the gains of both the position and velocity controller.

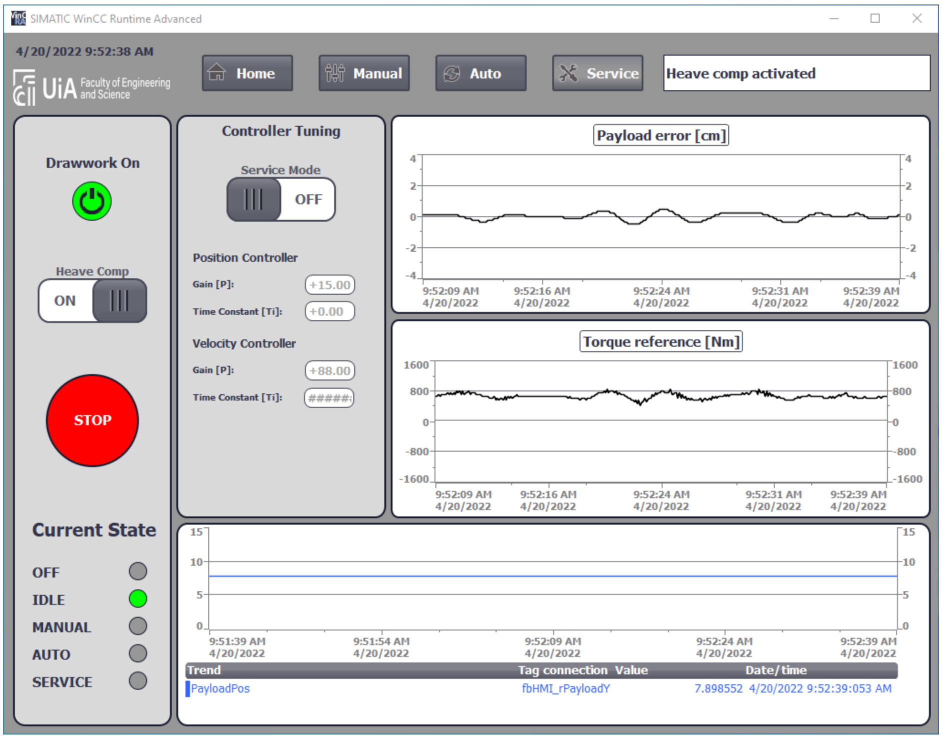


Figure 6.23: Operator HMI service page off.

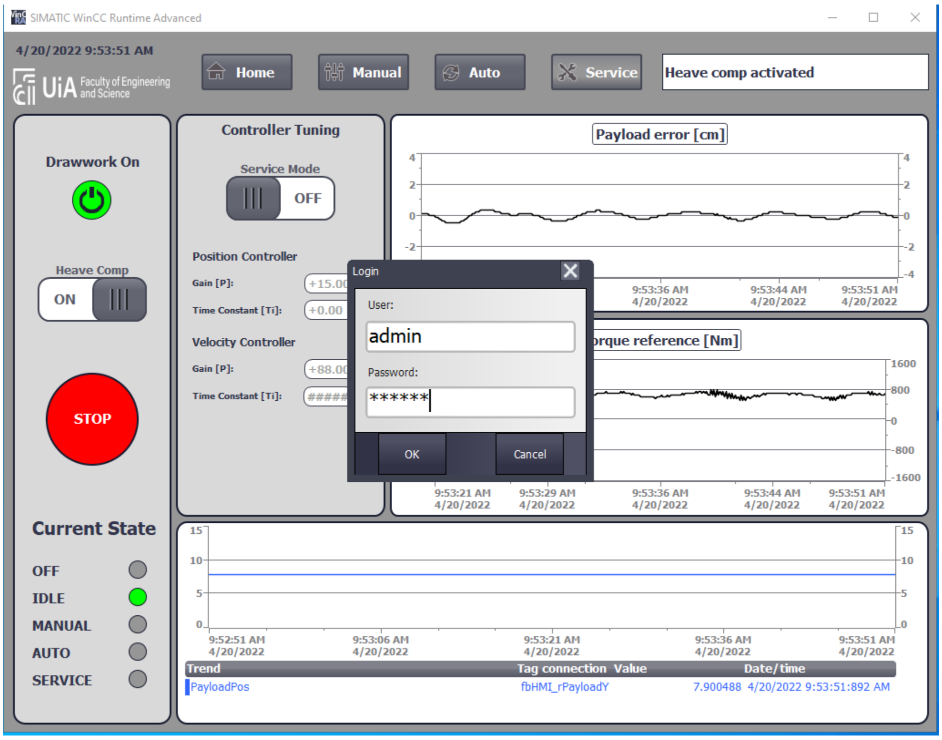


Figure 6.24: Operator HMI service page login.

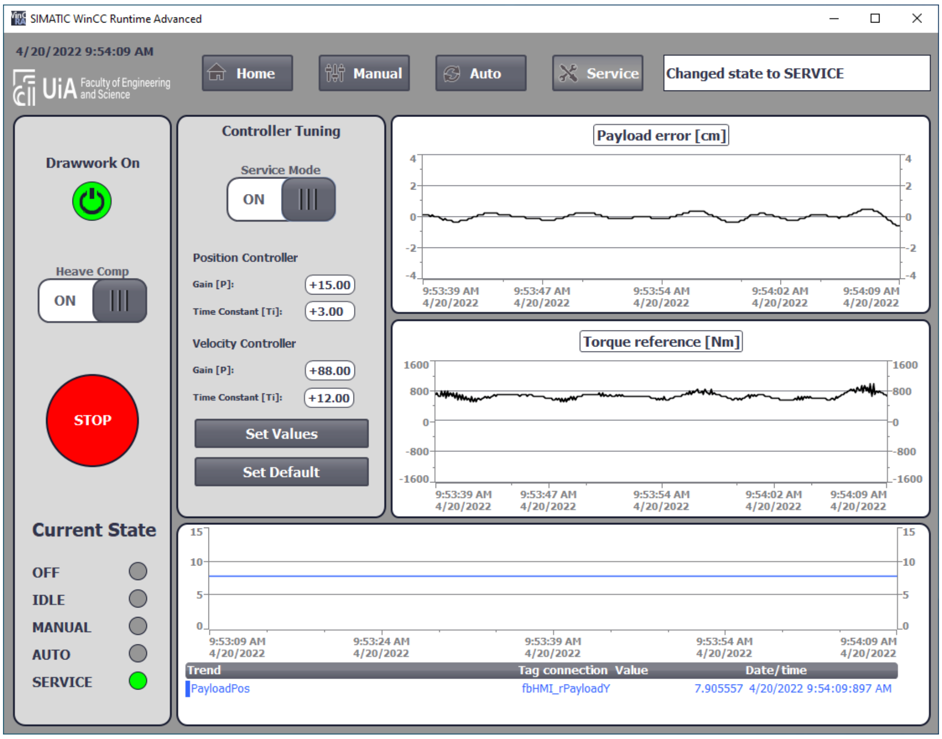


Figure 6.25: Operator HMI service page on.

6.6.2 Management HMI

The management HMI is not used by any operator, but rather the management, with the purpose of inspecting the current power usage, and energy usage since system start. In addition to this the management is able to see how many hours the system has been operational and vice versa. This information is essential to analyze the efficiency of the entire operation.

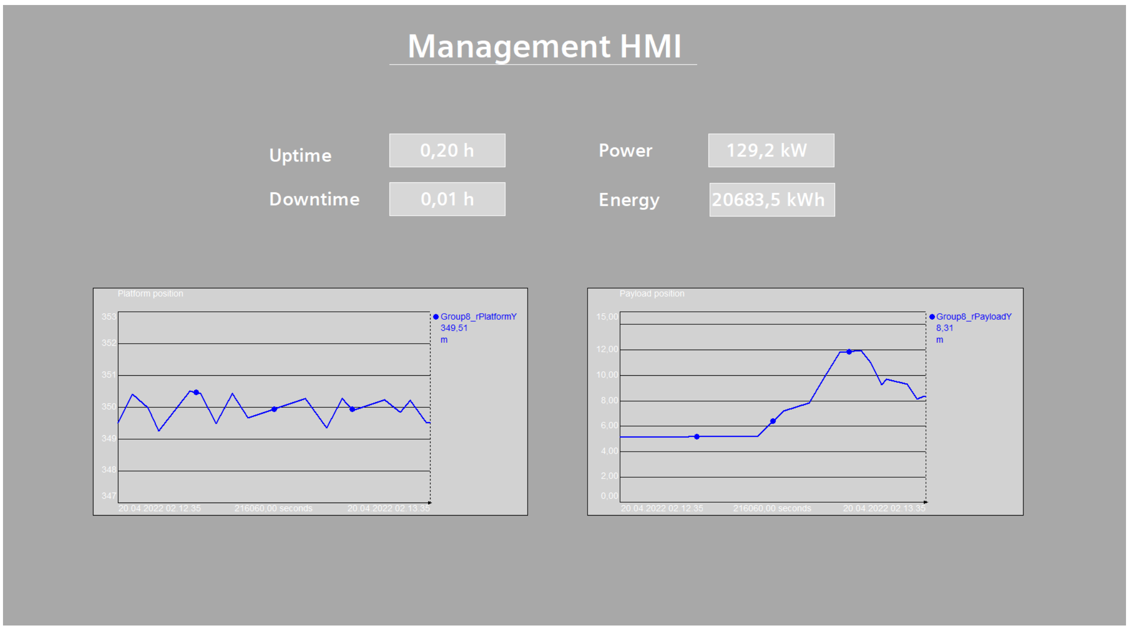


Figure 6.26: Management HMI.

7 Discussion

7.1 Project Management

In order to complete the project efficiently within the given time it is essential with a well organized project management. By having casual meetings alongside the formal supervised meetings the group has mostly been on top of the project. The Gantt chart has as well been a handy tool to ensure that every task is completed and for time-tracking of activities and deadlines.

By meeting the team regularly every member has known the struggles of any part of the project in any given time. That meant the members could help each other frequently. It also mean that even though the members had certain parts of the project to follow up, and attend to, they know the majority of what every part of the project concluded in.

With a morale of *"no such thing as a stupid question"* the group has been able to work efficient by easily and frequently asking each other for tips. That morale has also introduced the team to be comfortable with another, meaning the members being able to push each other as well as giving constructive feedback to improve the performance.

When following the Gantt chart and the overall plan of the project strictly, seeing when a part of the project could be closed, the group could easily take an overview of what was done. Followed by taking notes, and starting writing on the report early. In that manner when the project neared the end the group felt comfortable piecing the report together even though changes and improvements were made.

7.2 Drawwork

The dynamics of the drawwork is quite simple and therefore understandable to grasp. A basic foundation for how the dynamics should be implemented into the simulation model were thus laid. Firstly the simulation model were made using only the Simscape Driveline library consisting of translating- and rotating elements. Every parameter were added to the simulation model at the correct location with the given value, meaning for instance 5 sheaves, 4 ropes and multiple inertia's were implemented. However, the group quickly noticed that the solver in Simulink struggled to calculate the dynamics fast, hence the simulation were time consuming. In order to fix this it was decided to decompose the system as much as possible. That proved to be difficult because the Driveline library did not contain the preferred blocks. Parts of the model was therefore discarded and changed to consist of Simscape Multibody elements instead.

After finding a way to model the decomposed system using Multibody elements the simulation time were notably quicker which mean it will run more smoothly in real time on the Speedgoat. This is most likely due to fewer unknown variables and equations for the solver to solve. For instance instead of using the pulley elements provided which should be modelled as ideal, but each containing friction, inertia, mass, viscous damping and radius, the gear ratio provided from the sheaves were modelled into the gear instead. The same applies for the ropes which instead are modelled as a ideal infinite capacity spool of belt-cable.

When modelling using the Multibody library it is also possible to visualize the moving objects in the system. Which the group found was an interesting additional feature to the simulation.

The group is as well satisfied with the approach of how the platform motion created by the irregular waves were implemented. By switching between the seeds randomly and ensuring smooth transitions between the wave pattern meant the simulation model were tested using the wast majority of possible wave patterns. Meaning the controller is most likely tested in a worst scenario wave.

7.3 Electric Machines and Drives

The concept of an induction motor with field-oriented control were quite difficult to grasp. The concepts of how to control the motor and how the motor works are difficult. However, by spending enough time trying, failing and reading about it made it more understandable. That yielded in a dynamic motor model working within its performance limits. Using field-oriented control resulted in the motor working smoothly and being able to generate maximum torque at zero speed. In order to utilize the advantages of an induction motor in the simulation field-weakening and current saturation were implemented. Those made the motor being able to perform greater than its nominal performance limits. In the aftermath of this project, with the knowledge learned, a less powerful motor could perhaps be used instead because of field-weakening and current-saturation.

From the motor parameters is was attempted to find the correct equivalent circuit parameters using test setups for the motor. However, this proved to be quite the challenge. If the tests described in 4.2.3 is understood correctly they were completed inaccurately yielding circuit parameters that made the induction motor behave incorrectly. It has also been attempted to find the equivalent circuit parameters using the Particle Swarm Optimization Algorithm [4]. However that yielded the same results. In order to simulate the dynamic motor as closely as possible to how it would behave in the real world it was therefore chosen to use a calculator from Matlab to find them instead. Using that calculator gave equivalent circuit parameters that made the behave as intended.

7.4 Control System

The project description stated that the controller type had to be chosen between a position and velocity controller in cascade with additional feedforward, or a pure position controller with feedforward. A cascade controller where the inner loop is able to counteract disturbances faster than the outer, maybe even before the primary controller picks up the disturbance, seemed like the correct choice for an AHC system. The control structured was changed several times during the project period, however the final version where vertical motion is related to equivalent motor quantities open a broader specter of additional features that could be implemented. Examples of an additional feature coming from this conversion is the possibility to control the winch without AHC activated.

The control system as a whole consists of four PI controller. Where one of them is the current controller inside the electric drive, the latter three is the cascade loop and constant tension controller. Tuning was done using the Good Gain method, which worked very well. The gains had to be tuned manually by intuition during testing to work as intended for all possible scenarios. The final resulting control system showed great performance and accuracy with a maximum payload position error in the millimeter regions.

Further, having a complete mechanical simulation model with the motor and drive model implemented the exact way as it is on the real-time target was very beneficial. Especially with the control system sampled and used on the same discrete form as the PLC.

7.5 Industrial IT

Using hardware-in-the-loop simulation is a cost and time effective method to improve the development process, by increasing the amount of in-house development before implementing in the real world. In this project a simplified simulation model has been created in Simulink, and compiled to run on a real-time target. The main system logic is implemented on a Siemens PLC, which communicates with the real-time target using the TCP/IP protocol over Ethernet. Running the system logic in a state-machine proved to be a good choice, as it made the code more flexible for implementing new features along the development process.

Using industrial hardware such as a PLC, ensures that the logic is ran in real-time, which is optimal for implementing systems that require deterministic behavior. The control system designed in the Simulink-model was implemented directly on the Siemens PLC by using built-in function blocks. Which also provided several neat features such as output saturation, integral hold and the possibility to enable and disable the controller. To control the system, two human machine interfaces has been designed and implemented by using WinCC and ProcessBook.

The operator may use the prior to both manually and automatically control the payload position.

The automatic control allows the operator to generate a trajectory from the current position to a desired position using a fift-order polynomial. This ensures a smooth payload acceleration, which reduces the motor torque requirements. In ProcessBook the management can analyze the current power usage, and energy usage since the system startup. This makes it easier to make cost and time saving improvements to the operation.

8 Conclusion

In conclusion both the simulation models meets the design criteria well. The controller is designed and tuned yielding high accuracy with good performance. Based on tests performed on both simulation platforms the system has proven to be feasible giving expected motor performance results and being able to run in real-time. And can therefore be seen as a good indication on how the system realistically would behave. On the other side, by not being able to test the controller on a physical system, it is difficult to know how the FOC actually would behave. Further the drivetrain selection proved being able to perform the task. The PLC program and HMIs are designed according to the design specification set with additional features.

In final words the project has been fun, interesting and challenging, meaning it has required good and strategical project management to be completed efficiently.

Bibliography

[1] M. Marie Aftab M. Faisal; Choux. “Active Heave Compensation of Drawwork.” 2022. Not published.

[2] Henning Dierks. “PLC-automata: A new class of implementable real-time automata.” In: *Transformation-Based Reactive Systems Development*. Ed. by Miquel Bertran and Teodor Rus. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 111–125. ISBN: 978-3-540-69058-0.

[3] Kim R. Fowler. “Introduction to Good Development.” In: *Developing and Managing Embedded Systems and Products*. Ed. by Kim R. Fowler and Craig L. Silver. Oxford: Newnes, 2015, pp. 1–38. ISBN: 978-0-12-405879-8. DOI: https://doi.org/10.1016/B978-0-12-405879-8.00001-5. URL: https://www.sciencedirect.com/science/article/pii/B9780124058798000015.

[4] M. O. Gülbahçe and M. E. Karaaslan. “Estimation of Induction Motor Equivalent Circuit Parameters from Manufacturer’s Datasheet by Particle Swarm Optimization Algorithm for Variable Frequency Drives.” In: *Electrica* 22.1 (2022), pp. 16–26. DOI: 10.5152/electrica.2021.21122.

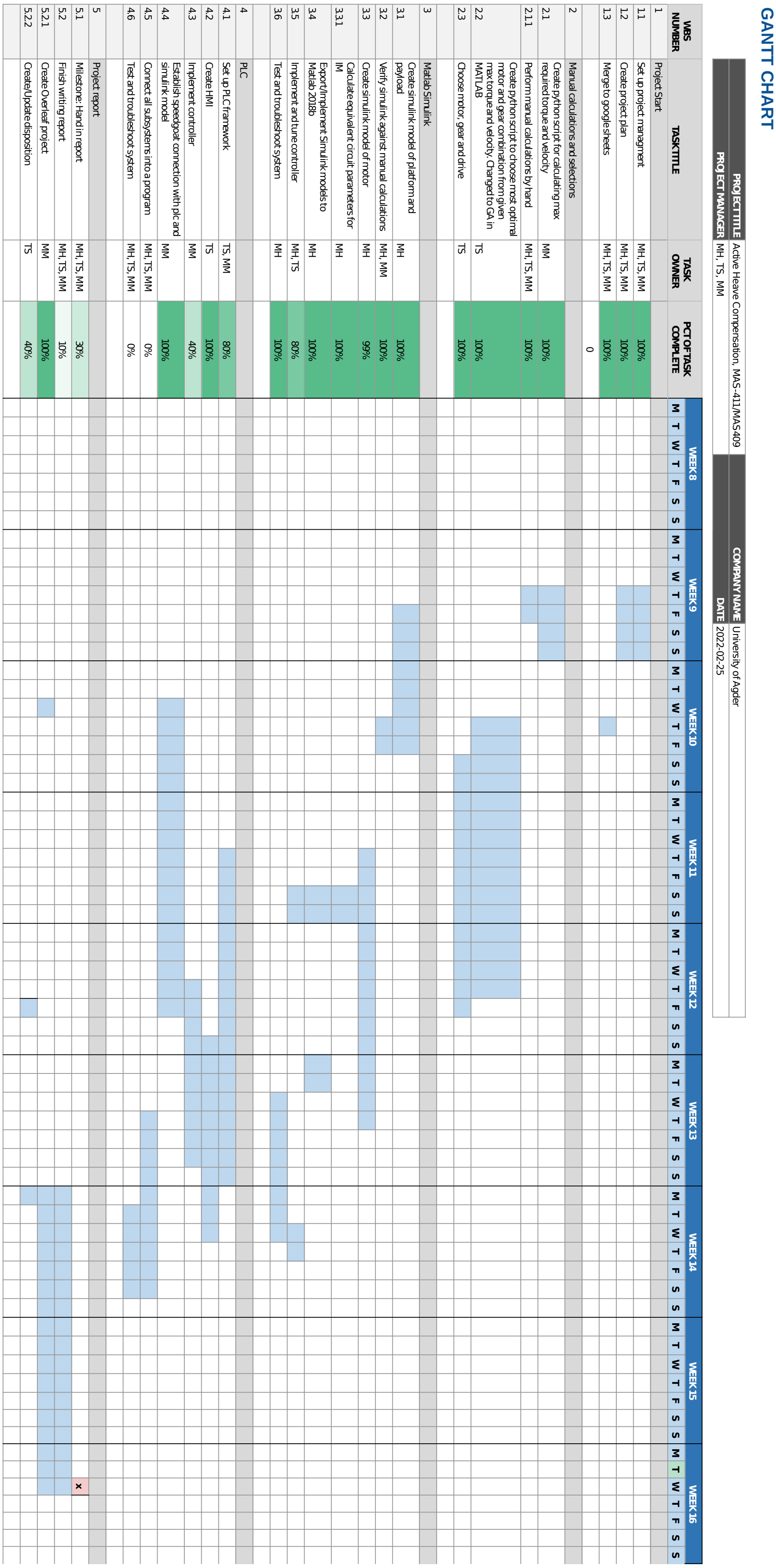
[5] Finn Haugen. “Derivation of a Discrete-Time Lowpass Filter.” In: (2008), p. 3.

[6] Finn Haugen. “The Good Gain method for PI (D) controller tuning.” In: *Tech Teach* (2010), pp. 1–7.

[7] Lennart Harnefors; Marko Hinkkanen; Oskar Wallmark; Alejandro Gòmez Yepes. “Control of Voltage-Source Converters and Variable-Speed Drives.” 2015. URL: https://docplayer.net/61055480-Control-of-voltage-source-converters-and-variable-speed-drives.html. Not published.

A Project Management

A.1 Gantt Chart



A.2 Meeting Agenda & Minutes of Meeting

220304 - Meeting Agenda

Location: C4 095
Time: 11:30-11:50
Date: 04.03.2022
Participants: Martin Mæland, Martin Hermansen, Muhammad Faisal Aftab
Meeting leader: Martin Mæland
Minutes taker: Martin Hermansen

1 - Overview of project plan (gantt chart)

- Link to plan: https://app.agantty.com/?locale=en/#/project/930355.
- More specific tasks should be made.

2 - Work done (this week)

- Setup project management software.
    - Agnetty
    - Google Drive
- Using Gitlab for version control/database
- Setup project plan.
    - See gantt chart.
    - Switch to google sheets?
- Started on manual calculations.
    - Wave generator complete.
        - Finding velocity and acceleration as well.
    - Created formulas to find motor torque.
    - Started Python script for calculations.

3 - Planned work (next week)

- Choose components.
- Begin to setup simulation model in MATLAB.

4 - Questions

- How long should the report be?
- Is it one report for each course?
- Viscous friction?

5 - Minutes of Meeting

- Hand-in moved to 19th of April.
- Bug in Agantty, every task moved one day.
- Motor not important in MAS411. Can model the drive as a transfer function.

Answer to questions:

- No limit on the pages.
- Combined report.
- Viscous friction used for MATLAB/Simulink/Simscape drum model.

220311 - Meeting Agenda

Location: D3 051
Time: 11:30-11:50
Date: 11.03.2022
Participants: Martin Mæland, Martin Hermansen, Tarjei Skotterud, Muhammad Faisal Aftab
Meeting leader: **Martin H**
Minutes taker: **Martin M**

1 - Overview of project plan (gantt chart)

- Link to gantt chart.

2 - Work done (this week)

- Changed gantt chart software.
- Finished manual calculations.
  - Created python script.
- Finished platform model in Simulink.
  - Applied forces.
  - Wave motion.
  - Seabed.
  - Infinite strong motor.
- Verified manual calculations and simulink model.
- Almost finished gear/motor selection script.
  - Create python script to choose most optimal motor and gear combination from given max torque and velocity.
- Started on testing speedgoat connection with plc and simulink model

3 - Planned work (next week)

- Create simulink model of motor
- Implement and tune controller

- Set up PLC framework
- Create HMI

4 - Questions

5.2 Seabed

The seabed is modeled as a spring-damper, see Fig. 4, with $k = 1.8 \cdot 10^6$ N/m and $b = 6.5 \cdot 1^5[\frac{N.s}{m^{1.5}}]$

The force from the seabed on the payload can be modeled as a function of penetration and penetration rate:

$$F = \begin{cases} 0 & \Delta x < 0 \\ k.\Delta x + b\Delta\dot{x}\sqrt{\Delta x} & \Delta x \geq 0 \end{cases} \tag{2}$$

- b = 6.5 * 10^5 or 6.5 * 1^5?
- What about the unit?
- Can we use multibody simscape library?
    - Difficult to simulate rope. Pushing payload down into seabed.

5 - Minutes of Meeting

- Unit: 10^5, and only m/s not m/s^1.5
-

220318 - Meeting Agenda

Location: D3 051
Time: 11:30-11:50
Date: 2022-03-18
Participants: Martin Mæland, Martin Hermansen, Tarjei Skotterud, Muhammad Faisal Aftab
Meeting leader: Tarjei Skotterud
Minutes taker: Martin Mæland

1 - Overview of project plan (gantt chart)

- Link to gantt chart.

2 - Work done (this week)

- Finished gear/motor selection script.
  - Generates 100 random waves and checks max and min velocities and accelerations.
  - Generates trajectories from -8m/s lowering to -1m/s lowering with 0.1m/s increments.
  - Calculates max required motor torque and velocity at that point.
  - Calculates max required velocity and torque at that point.
  - Checks every motor from the catalog with every gear ratio (1-10) and outputs every possible combination according to the torques and velocities.
  - Easily comparable to cost and optimal gear ratio.
  - Planning to plot the cost versus velocity (of payload), from which a motor and gear is chosen.
- Tried to get speedgoat up and running.
  - Having some connectivity issues with MATLAB versions newer than 2019.
  - Having compiling issues with MATLAB versions older than 2020.
- Project report disposition.
- Started working with motor simulation.
  - Difficult to grasp

3 - Planned work (next week)

- Motor simulation in Simulink (hopefully this week).
- Finish speedgoat setup.
- Set up PLC framework and begin on HMI.

- Implement a cascade controller.

4 - Questions

- Having some issues with motor and gear selection due to high torque and velocity requirements from the AHC. Estimating that no motor and gear combination would work while lowering, even though quintic interpolation is used to reduce motor torque. Is there something in the project requirements that should be tweaked? Eg. increase the wave period.

5 - Minutes of Meeting

- Speedgoat Matlab version 2018b.
- Using Quintic interpolation to generate the path/trajectory.

220325 - Meeting Agenda

Location: D3 051
Time: 11:30-11:50
Date: 2022-03-25
Participants: Martin Mæland, Martin Hermansen, Tarjei Skotterud, Muhammad Faisal Aftab
Meeting leader: Martin Mæland
Minutes taker: Tarjei Skotterud

1 - Overview of project plan (gantt chart)

- Link to gantt chart.

2 - Work done (this week)

- Discarded the brute force python calculation for drivetrain selection and did a cost+dynamics optimization using GA in Matlab to select most suitable components.
- Motor, gear and drive have been selected.
- Working motor model, but needs some tuning.
- Speedgoat and PLC communication established. Tested and verified data and different sample rates with simple sine waves back and forth. Currently at 10ms.
- Started on PLC system structure: link

3 - Planned work (next week)

- Finish motor simulation.
- Combine motor simulation with platform/payload simulation.
- Implement PLC system structure.
- Create HMI (maybe postponed to week 14)

4 - Questions

- Is it correct to model the induction motor from scratch?

5 - Minutes of Meeting

- Ask Martin Choux about induction motor model.
-

220401 - Meeting Agenda

Location: D3 051
Time: 11:30-11:50
Date: 2022-04-01
Participants: Martin Hermansen, Tarjei Skotterud, Muhammad Faisal Aftab
Meeting leader: Martin Hermansen
Minutes taker: Tarjei Skotterud

1 - Overview of project plan (gantt chart)

- Link to gantt chart.

2 - Work done (this week)

- Simulink part more or less completed.
    - Converted to Matlab 2018b
    - Platform model
    - IM model
    - Combined motor model with platform simulation.
- HMI is well underway, and will be finished this week.
- More or less finished PLC system structure: link.
- Started implementing PLC system structure into TiaPortal.

3 - Planned work (next week)

- Wrap up all subsystems.
    - Merge HMI tags to PLC
    - PLC system structure
    - Speedgoat - Simulink
- Combine all subsystems into a working program.
    - Tune system.
    - Finish program(?).
- Modify report disposition to match work-flow after having a working program.
- Write on the report and document essential material.

4 - Questions

- HMI Questions

- Tuning

5 - Minutes of Meeting

- Tune in Matlab, keep it discrete.
- Error in matlab. Have 1 sample time delay.

220408 - Meeting Agenda

Location: D3 051
Time: 11:30-11:50
Date: 2022-04-08
Participants: Martin Hermansen, Tarjei Skotterud, Martin Mæland Muhammad Faisal Aftab
Meeting leader: **Martin Mæland**
Minutes taker: Martin Hermansen

1 - Overview of project plan (gantt chart)

- Link to gantt chart.

2 - Work done (this week)

- Finished HMI
- Finished PLC
- Merging subsystems
- Updated report disposition

3 - Planned work (next week)

- Troubleshoot and improve system
- Write on the report and document essential material.

4 - Questions

- Power usage KPI in management HMI; delivered electrical power or output mechanical?

5 - Minutes of Meeting

-

B MATLAB Scripts

B.1 WaveGenerator function

```
function [ydd,y,yd] = WaveGenerator(t_sim,T_sw,T_seed,seed)

    Tw = 10;     % mean period [s]
    Hs = 1.7;    % significant wave height

    w = linspace(0.1,2.0,30);    % frequency range
    delta_w = w(2)-w(1);

    % PM Spectrum (frequency domain)
    w1 = (2*pi)/Tw;
    A = 0.11*(Hs^2)*(w1^4);
    B = 0.44*(w1^4);
    S = (A./w.^5).*exp(-(B./(w.^4)));

    A_wave = sqrt(2*S.*delta_w);     % amplitude of each sine wave

    t = t_sim - T_seed*double(seed); % relative time variable [0, T_sw]

    if seed ~= 0

        % phaseshifts from previous seed
        rng(seed-1);
        phi_prev = 2*pi*(rand(1,length(w))-0.5);

        % phaseshift with new seed
        rng(seed);
        phi = 2*pi*(rand(1,length(w))-0.5);

        % generate trajectory to interpolate on to the new wave
        if t < T_sw

            % find initial- and end point in previous and
            % new wave respectively
            wave_prev = WavePoint(T_seed,A_wave,w,phi_prev);
            wave = WavePoint(T_sw,A_wave,w,phi);

            % calculate polynomial coefficients with wavepoints
            % as trajectory constraints
            a = Quintic(...
                0,T_sw,...
                wave_prev(1),wave(1),...
                wave_prev(2),wave(2),...
                wave_prev(3),wave(3)...
            );

            % interpolate
            y = a(1)+a(2)*t+a(3)*t^2+a(4)*t^3+a(5)*t^4+a(6)*t^5;
            yd = a(2)+2*a(3)*t+3*a(4)*t^2+4*a(5)*t^3+5*a(6)*t^4;
            ydd = 2*a(3)+6*a(4)*t+12*a(5)*t^2+20*a(6)*t^3;
        else
            % find current point in wave
            wave = WavePoint(t,A_wave,w,phi);

            % output wave
            y = wave(1);
            yd = wave(2);
            ydd = wave(3);
```

```
        end
    else
        % same as above but for the initial wave where it should be
        % interpolated from zero
        rng(seed);
        phi = 2*pi*(rand(1,length(w))-0.5);

        if t < T_sw
            wave = WavePoint(T_sw,A_wave,w,phi);

            a = Quintic(...
                0,T_sw,...
                0,wave(1),...
                0,wave(2),...
                0,wave(3)...
            );

            y = a(1)+a(2)*t+a(3)*t^2+a(4)*t^3+a(5)*t^4+a(6)*t^5;
            yd = a(2)+2*a(3)*t+3*a(4)*t^2+4*a(5)*t^3+5*a(6)*t^4;
            ydd = 2*a(3)+6*a(4)*t+12*a(5)*t^2+20*a(6)*t^3;
        else
            wave = WavePoint(t,A_wave,w,phi);

            y = wave(1);
            yd = wave(2);
            ydd = wave(3);
        end
    end


    function a = Quintic(t0,t1,p0,p1,v0,v1,a0,a1)
        % time constraints
        matA = [
                1, t0, t0^2, t0^3, t0^4, t0^5;
                1, t1, t1^2, t1^3, t1^4, t1^5;
                0, 1, 2*t0, 3*t0^2, 4*t0^3, 5*t0^4;
                0, 1, 2*t1, 3*t1^2, 4*t1^3, 5*t1^4;
                0, 0, 2, 6*t0, 12*t0^2, 20*t0^3;
                0, 0, 2, 6*t1, 12*t1^2, 20*t1^3
        ];

        % path constraints
        vecB = [p0;p1;v0;v1;a0;a1];

        % output Quintic coefficients
        a = matA\vecB;
    end

    function wave = WavePoint(t,A_wave,w,phi)
        wave = zeros(3);
        wave(1) = sum(A_wave.*cos(w*t+phi));            % position
        wave(2) = sum(-A_wave.*w.*sin(w*t+phi));        % velocity
        wave(3) = sum(-A_wave.*(w.^2).*cos(w*t+phi));   % acceleration
    end
end
```

B.2 Max and Min Wave Acceleration

```
clc; clear; close all;

Hs = 1.7;
Tw = 10;
s_fr = 100;
s_T = 60;

point = zeros(4,3);


for seed = 0:1000

    [p,v,a,t] = PMSpectrum(Hs,Tw,s_fr,s_T,seed);

    [v_min,i] = min(v);
    if v_min < point(1,2)
        point(1,:) = [p(i),v(i),a(i)];
    end

    [v_max,i] = max(v);
    if v_max > point(2,2)
        point(2,:) = [p(i),v(i),a(i)];
    end

    [a_min,i] = min(a);
    if a_min < point(3,3)
        point(3,:) = [p(i),v(i),a(i)];
    end

    [a_max,i] = max(a);
    if a_max > point(4,3)
        point(4,:) = [p(i),v(i),a(i)];
    end

end

disp('      pos       vel       acc');
disp(' ');
disp(point);
```

C Drivetrain Selection

C.1 Main

```
global y_pl_t y_pl_tMax y_pl_tt y_t y_tt r_D i_sh Fg Fb rho Cd A_pl mL g J_D

g = 9.81;
A_pl = 1.5;
mL = 12600;
J_D = 1;
Cd = 1.8;
rho = 1027;
Fg = mL*g;
Fb = rho*g*2;
i_sh = 4;
r_D = 0.2323/2;

y_t = 1.1364;
y_tt = -1.0627;

%   gear motor drive
lb = [1    1   1];
ub = [91 31 23];

Tp = 10;

[p,v,a,t] = Quintic(0.01,0,Tp,7.5,5,0,0,0,0);

y_pl_tMax = 0;%min(v)
y_pl_tt = 0;%max(a)
y_pl_t = 0;%v(i)

opts = optimoptions(@ga, ...
                    'PopulationSize', 150, ...
                    'MaxGenerations', 200, ...
                    'EliteCount', 10, ...
                    'FunctionTolerance', 1e-8, ...
                    'PlotFcn', @gaplotbestf);


rng(0, 'twister');
[xbest, fbest, exitflag] = ga(@DrivetrainCost, 3, [], [], [], [], ...
    lb, ub, @DrivetrainConstraints, [1 2 3],opts);

display(xbest);

fprintf('\nCost function returned by ga = %g\n', fbest);

[cost, j_match] = CalcResults(xbest);
fprintf('\nTotal cost: %g\n', cost);
fprintf('Inertia matching (Jm/Jeq): %g\n', j_match);
```

C.2 Calculate Results

```
function [cost, J_match] = CalcResults(x)

    global y_pl_t y_pl_tt r_D i_sh Fg Fb mL g J_D rho Cd A_pl

    % Map the discrete variables
    [gear,motor,drive] = DrivetrainData(x);

```

```
    gb = gear(1);
    C1 = gear(2);
    C2 = motor(12);
    C3 = drive(12);
    Jm = motor(10);

    % drag force
    Fd = (rho*y_pl_t^2*Cd*A_pl)/2;

    % equivalent inertia
    J_eq = (J_D+(mL*y_pl_tt-Fb+Fg+sign(y_pl_t)*Fd)/i_sh*r_D^2/g)/(gb^2);

    cost = C1+C2+C3;
    J_match = Jm/J_eq;

end
```

C.3 Drive Cost

```
function cost = CostDrive(Pc)
    % calculates the cost of a drive based on its power

    W_c = 2.5;
    Pc_max = 200;

    cost = W_c*(1 + Pc/Pc_max);

end
```

C.4 Gearbox Cost

```
function cost = CostGB(R_GB)
    % calculates the cost of a gearbox given the gearbox ratio as an input

    W_GB = 2;

    cost = W_GB*(1+R_GB/10);
end
```

C.5 Motor Cost

```
function cost = CostMotor(Pm, np)
    % calculates the cost of a motor given the nominal power
    % and number of poles

    W_M = 2;
    Pm_max = 200;

    cost = W_M*(1 + Pm/Pm_max + abs(np-4)/4);
end
```

C.6 Drivetrain Constraints

```
function [c, ceq] = DrivetrainConstraints(x)

    %   [c, ceq] = DriveTrainConstraints(x) calculates the constraints on the
    %   component for Solving a Mixed Integer Engineering Design
    %   Problem Using the Genetic Algorithm applied to drive train component
    %   selection.

```

```
    global y_pl_t y_pl_tMax y_pl_tt y_t y_tt r_D i_sh Fg Fb rho Cd A_pl mL ...
   g J_D

    % Map the discrete variables
    [gear,motor,drive] = DrivetrainData(x);
    gb = gear(1);
    n_p = motor(11);
    T_N = motor(7);
    P_N = motor(2)*1e3;
    I_N = motor(5);
    n_N = motor(3);
    PF = motor(4);
    Jm = motor(10);
    T_ratio = motor(9);
    I_d = drive(2);

    % calculations
    v_max = abs(y_pl_tMax - y_t);
    a_max = abs(y_pl_tt - y_tt);

    omega_max = v_max*i_sh*gb/r_D;
    n_max = omega_max*60/2/pi;
    F_wire = (Fg-Fb)/i_sh;
    T_cont = F_wire*r_D/gb;

    Fd = (rho*y_pl_t^2*Cd*A_pl)/2;
    F_wire_acc = (mL*y_pl_tt-Fb+Fg+sign(y_pl_t)*Fd)/i_sh;

    alfa_max = a_max*i_sh*gb/r_D;

    J_load = (J_D + F_wire_acc*r_D^2/g)/(gb^2);

    T_acc = alfa_max*(Jm + J_load);
    T_max = T_cont + T_acc;

    P_cont = abs(T_cont*y_t*i_sh*gb/r_D);

    phi = acos(PF);
    n_ratio = max([1 n_max/n_N]);
    i_sd = ...
   I_N*(n_ratio*(sin(phi)+cos(phi)*sqrt((T_ratio)^2-1))-cos(phi)*sqrt((T_ratio*n_ratio)
   -(min([T_ratio (T_cont/T_N)])/n_ratio)^2));
    i_sq = I_N*(T_cont/T_N*n_ratio)*cos(phi);
    i_m = sqrt(i_sq^2 + i_sd^2);

    c_torque = T_max - T_ratio*T_N;

    c_loadability = T_cont - Loadability(gb, n_p)*T_N;

    c_power = P_cont - P_N;

    c_current = i_m - I_d;

    c = [c_torque; c_loadability; c_power; c_current];

    ceq = [];
end
```

C.7 Drivetrain Cost

```
function C = DrivetrainCost(x)
```

```

    global y_pl_t y_pl_tt r_D i_sh Fg Fb mL g J_D rho Cd A_pl

    % Map the discrete variables
    [gear,motor,drive] = DrivetrainData(x);

    gb = gear(1);
    C1 = gear(2);
    C2 = motor(12);
    C3 = drive(12);
    Jm = motor(10);

    % total median cost of each component together
    mean_cost = 3.1+4.1+3.2;
    Fd = (rho*y_pl_t^2*Cd*A_pl)/2;

    J_eq = (J_D+(mL*y_pl_tt-Fb+Fg+sign(y_pl_t)*Fd)/i_sh*r_D^2/g)/(gb^2);
    C = (C1+C2+C3)/mean_cost;% + 0.1*abs(Jm/J_eq -1);
end
```

C.8 Drivetrain Data

```
function [gb,motor,drive] = DrivetrainData(x)

    % map integer variables to a discrete set


    % gear ratio vector of 1 to 10 with 0.1 steps (total of 91 options)
    r_gb = (1:0.1:10)';

    % possible values for x(1)
    %   gb  C1
    allX1 = [r_gb CostGB(r_gb)];


    % load catalog motordata from csv file
    motors = readmatrix('motor-drive_data/motor_data_matlab.csv');

    % possible values for x(2)
    %     1         2     3    4     5     6     7      8       9     10   11  12
    % catalog_nr  P_N   n_N  PF  I_N  I_S  T_N  Tl/Tn  Tb/Tn  Jm   np  C2
    allX2 = [motors CostMotor(motors(:,2), motors(:,11))];


    % load catalog motordata from csv file
    drives = readmatrix('motor-drive_data/drive_data.csv');

    % possible values for x(3)
    %     1         2     3      4     5      6      7      8     9    10    11   12
    % catalog_nr  I_N  I_max  P_N  I_ld  P_ld  I_hd  P_hd  DB  heat  air  C3
    allX3 = [drives CostDrive(drives(:,4))];


    % Map x(1), x(2), x(3) from the integer values used by GA to the
    % discrete values required.
    gb = allX1(x(1),:);
    motor = allX2(x(2),:);
    drive = allX3(x(3),:);
end
```

C.9 Loadability

```
function [res] = Loadability(gb,n_p)
global r_D y_pl_tMax y_t i_sh


% loadability curve:
x = [0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100]; % [Hz]
y = [0.80 0.100 0.100 0.100 0.100 0.100 0.100 0.100 0.100 0.100 0.92 0.84 ...
    0.76 0.68 0.63 0.57 0.53 0.49 0.46 0.43 0.42]; % [-]

v_min = 0;
v_max = abs(y_pl_tMax - y_t);

% Calculations
omega_max = v_max*i_sh*gb/r_D;
omega_min = v_min*i_sh*gb/r_D;
f_max = n_p*omega_max/2/pi;
f_min = n_p*omega_min/2/pi;

res = min([interp1(x,y,f_min) interp1(x,y,f_max)]);
end
```

C.10 Mean Cost

```

clc; clear; close all;

% gear ratio vector of 1 to 10 with 0.01 steps (total of 901 options)
r_gb = (1:0.01:10)';

% possible values for x(1)
%   gb  C1
allX1 = [r_gb CostGB(r_gb)];


% load catalog motordata from csv file
motors = readmatrix('motor-drive_data/motor_data_matlab.csv');

% possible values for x(2)
%     1         2     3    4     5     6     7      8       9     10   11 12
% catalog_nr  P_N   n_N  PF   I_N   I_S   T_N   Tl/Tn  Tb/Tn   Jm   np C2
allX2 = [motors CostMotor(motors(:,2), motors(:,11))];


% load catalog motordata from csv file
drives = readmatrix('motor-drive_data/drive_data.csv');

% possible values for x(3)
%     1         2     3      4     5      6      7      8     9    10    11   12
% catalog_nr  I_N  I_max  P_N  I_ld  P_ld  I_hd  P_hd  DB  heat  air C3
allX3 = [drives CostDrive(drives(:,4))];


mean(allX1(:,2))
mean(allX2(:,12))
mean(allX3(:,12))
```

D Configurations

D.1 Simulink/Simscape solver settings

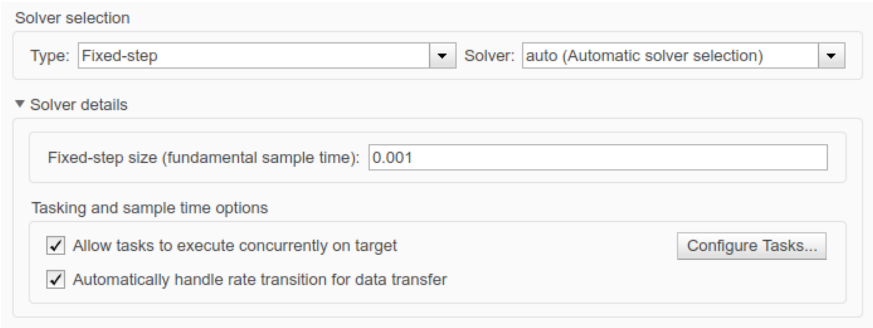


Figure D.1: Simulink solver configuration for ahc_409 model

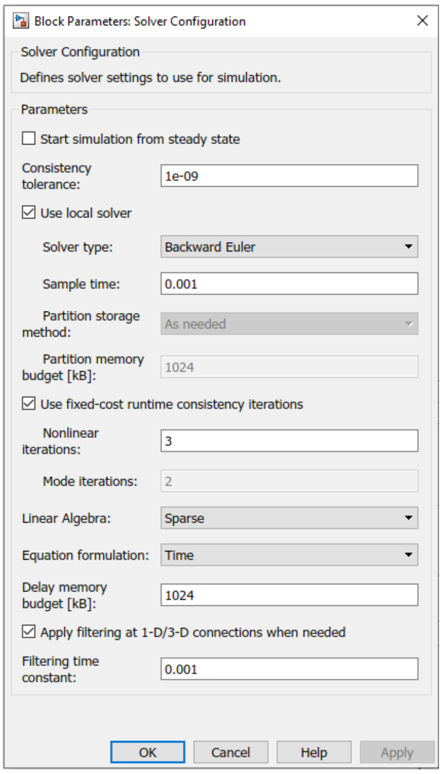
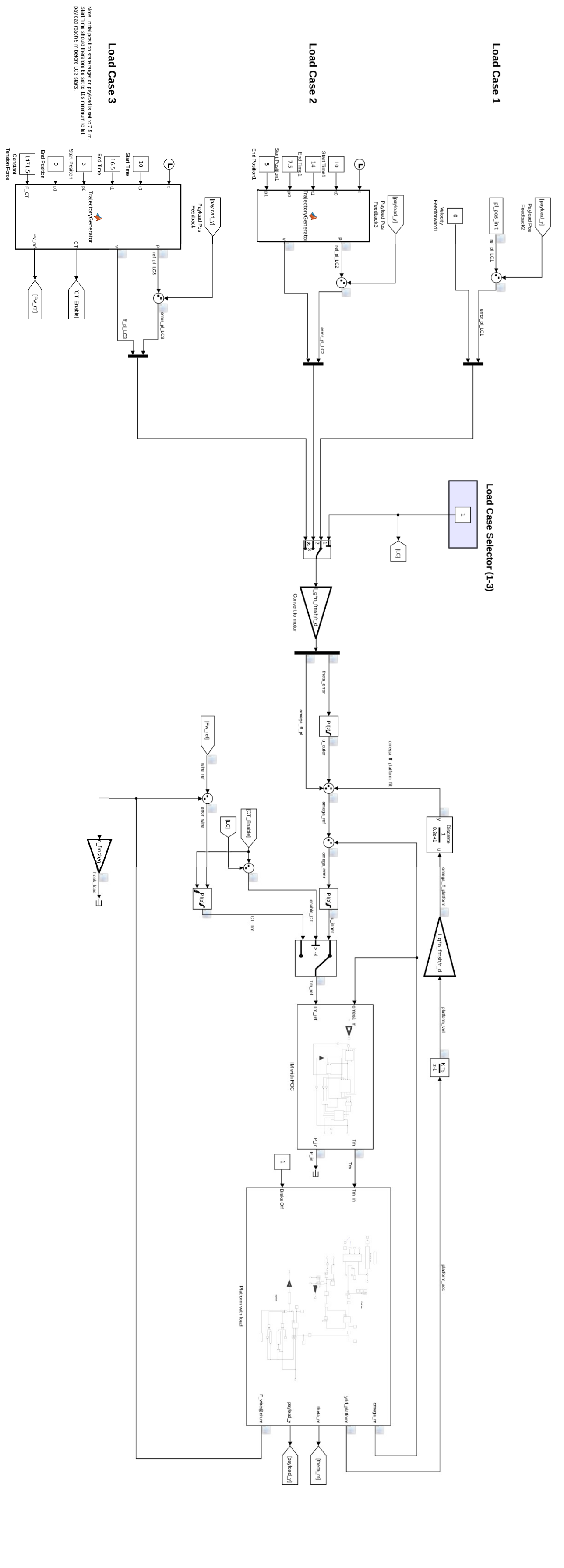


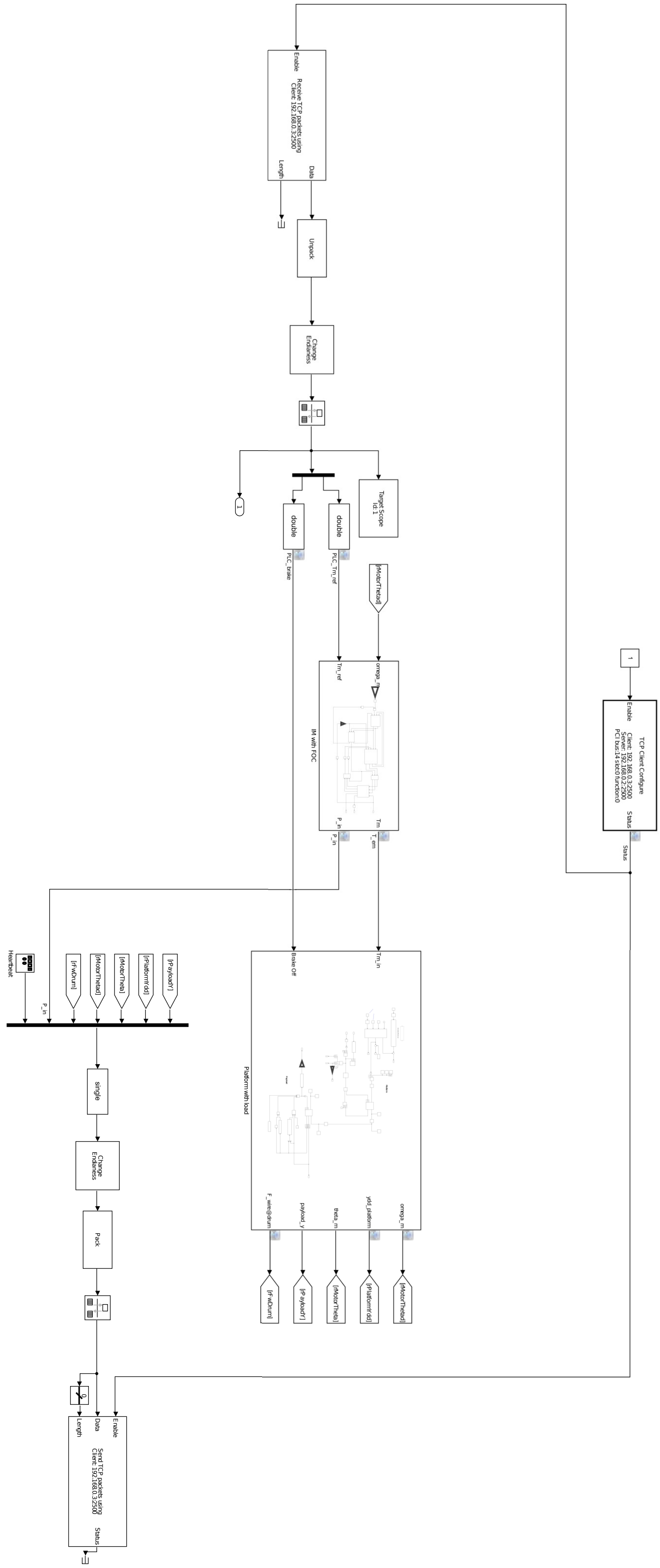
Figure D.2: Simscape solver configuration

E Simulink

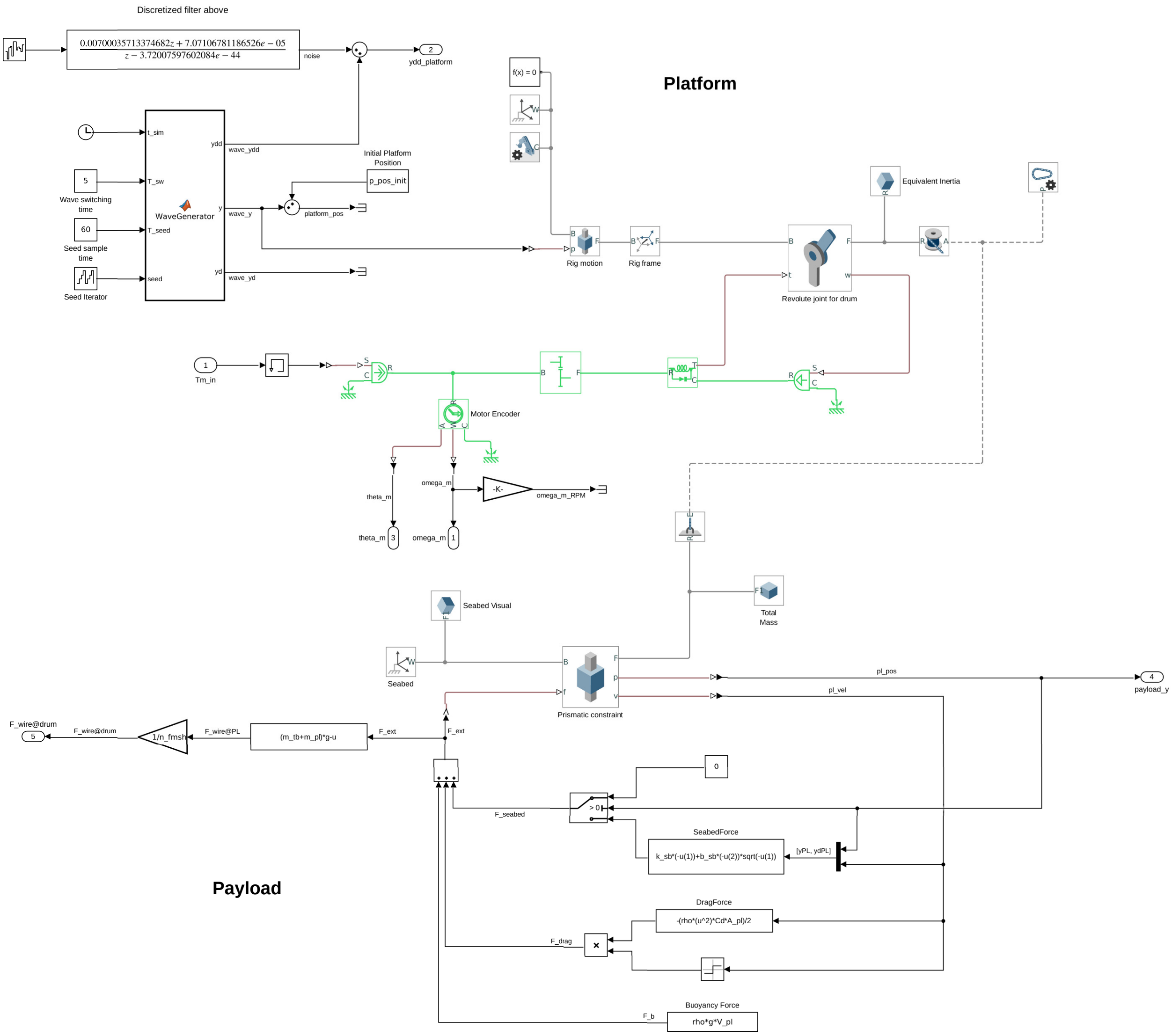
E.1 MAS409 Simulation Model



E.2 MAS411 Simulation Model

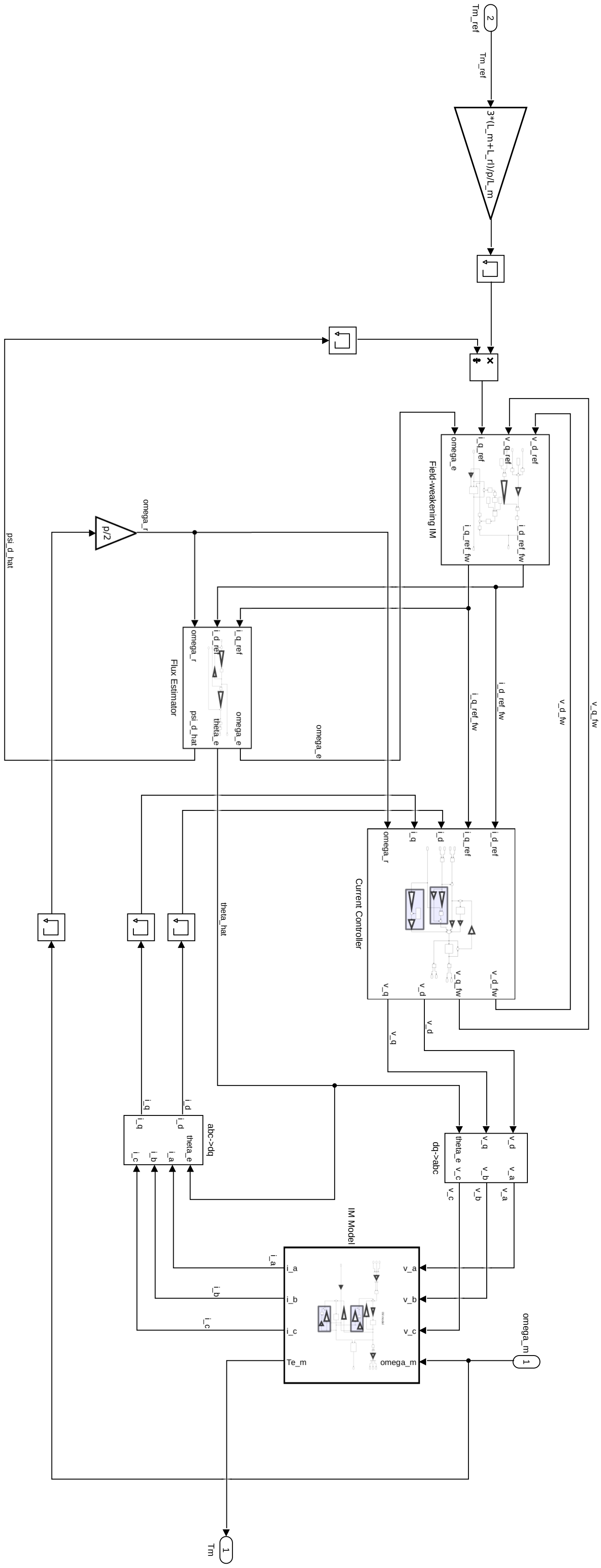


E.3 Drawwork

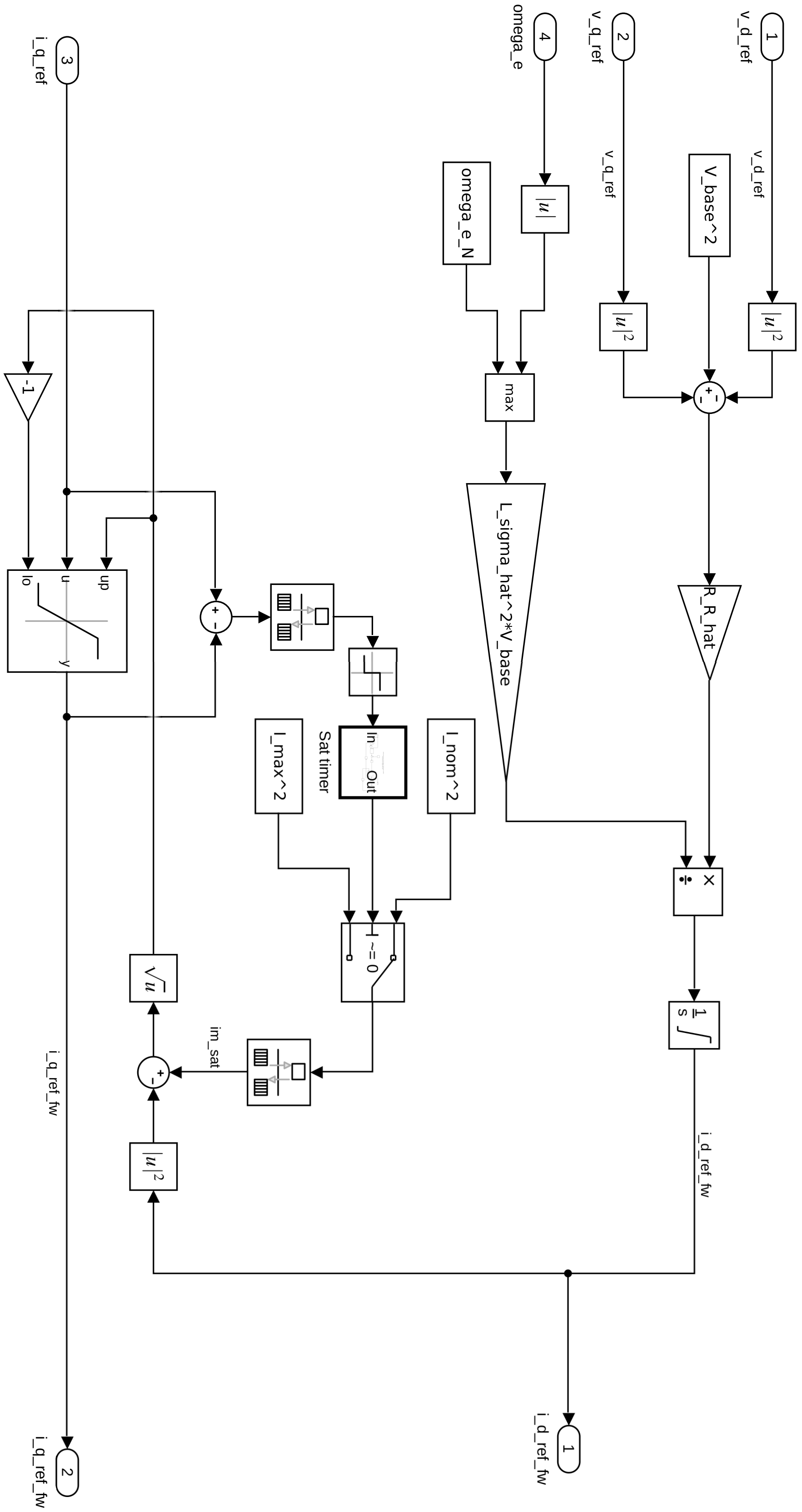


Discretized filter above
noise
ydd_platform
Platform
f(x) = 0
t_sim
T_sw
T_seed
seed
ydd
wave_ydd
y
wave_y
yd
wave_yd
WaveGenerator
5
Wave switching time
60
Seed sample time
Seed Iterator
Initial Platform Position
p_pos_init
platform_pos
Rig motion
Rig frame
Revolute joint for drum
Equivalent Inertia
Tm_in
Motor Encoder
theta_m
omega_m
omega_m_RPM
-K-
Seabed Visual
Seabed
Prismatic constraint
Total Mass
pl_pos
pl_vel
payload_y
F_wire@drum
F_wire@PL
1/n_fmsh
(m_tb+m_pl)*g-u
F_ext
F_seabed
SeabedForce
k_sb*(-u(1))+b_sb*(-u(2))*sqrt(-u(1))
[yPL, ydPL]
DragForce
-(rho*(u^2)*Cd*A_pl)/2
F_drag
Buoyancy Force
rho*g*V_pl
F_b
Payload

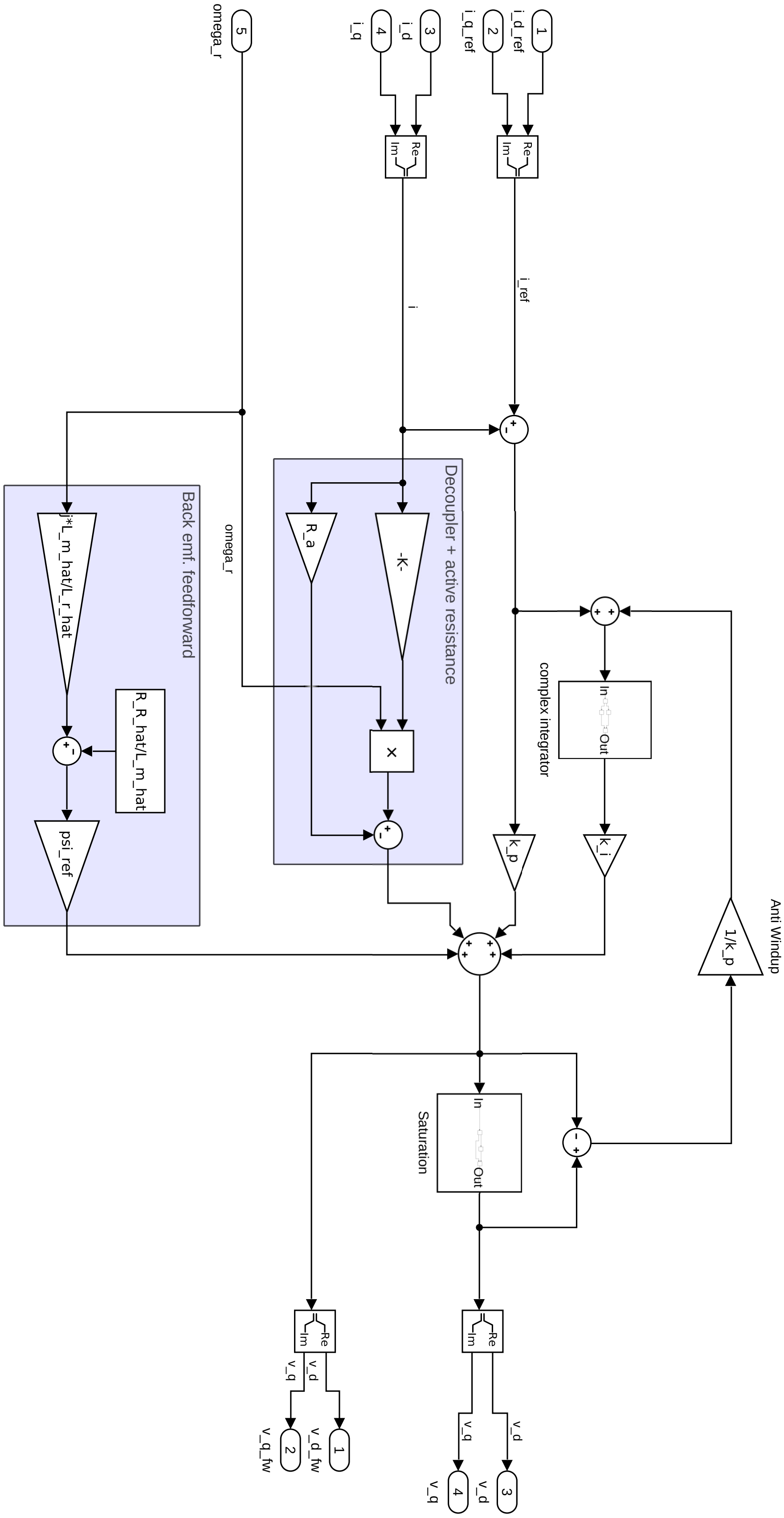
E.4 IM Motor with FOC Model



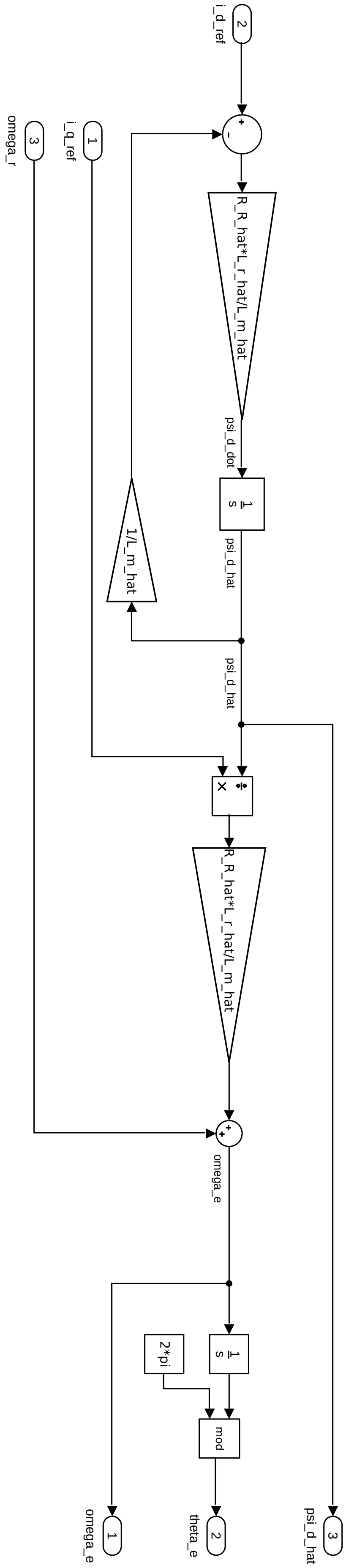
E.5 Field-Weakening



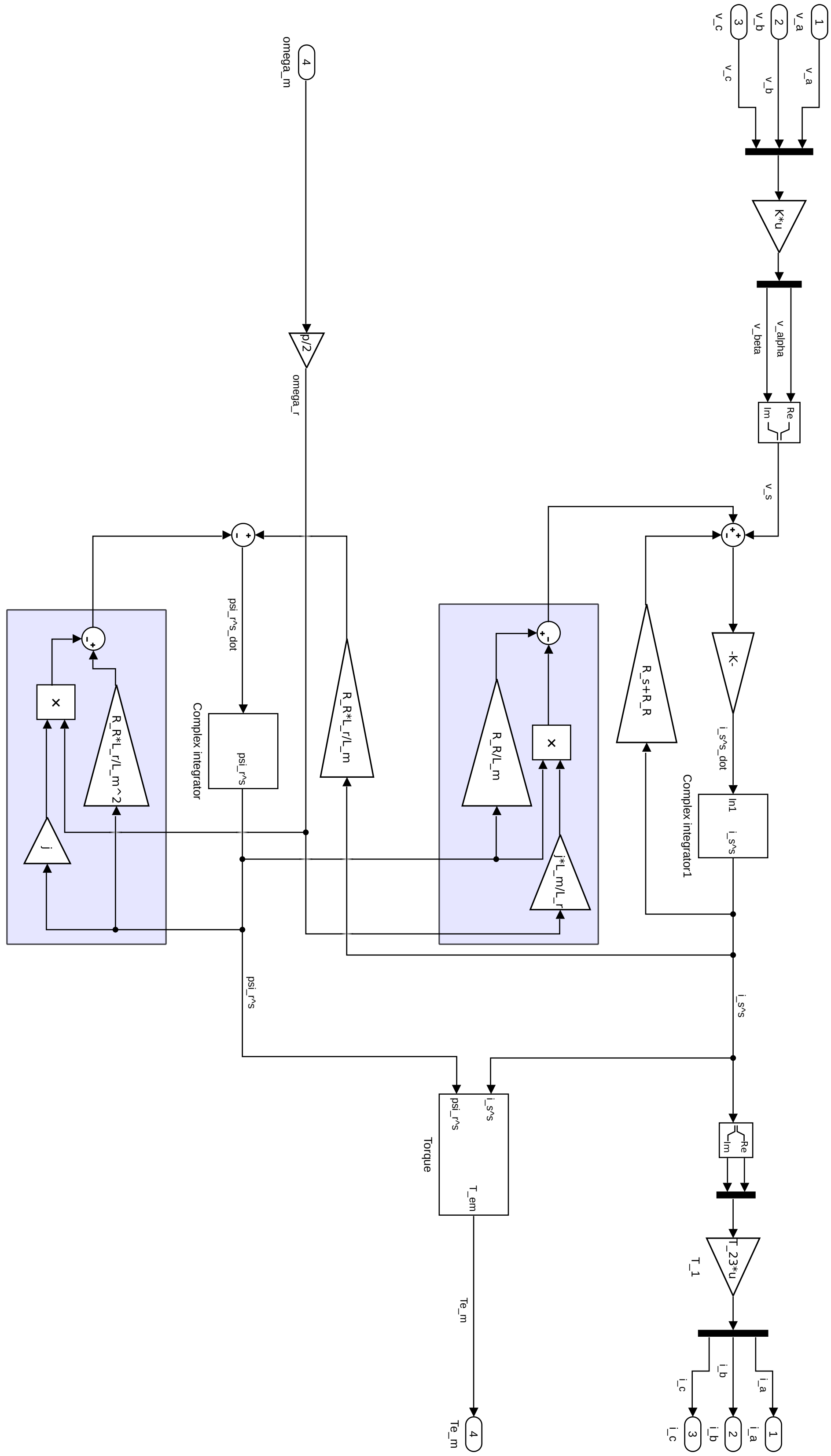
E.6 Current Controller



E.7 Flux Estimator



E.8 IM Model



v_a
v_b
v_c
K*u
v_alpha
v_beta
Re
Im
v_s
-K-
i_s^s_dot
In1
i_s^s
Complex integrator1
R_s+R_R
R_R/L_m
j*L_m/L_r
omega_m
p/2
omega_r
R_R*L_r/L_m
psi_r^s_dot
psi_r^s
Complex integrator
R_R*L_r/L_m^2
j
i_s^s
Torque
T_em
Te_m
T_23*u
T_1
i_a
i_b
i_c

E.9 Motor Test Setup

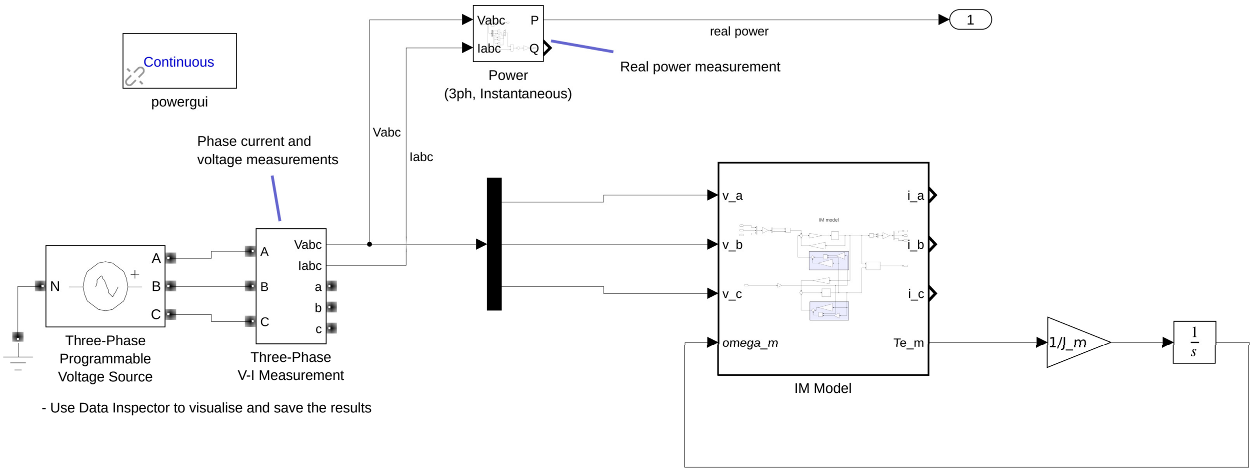


Figure E.1: Maximum Velocity setup.

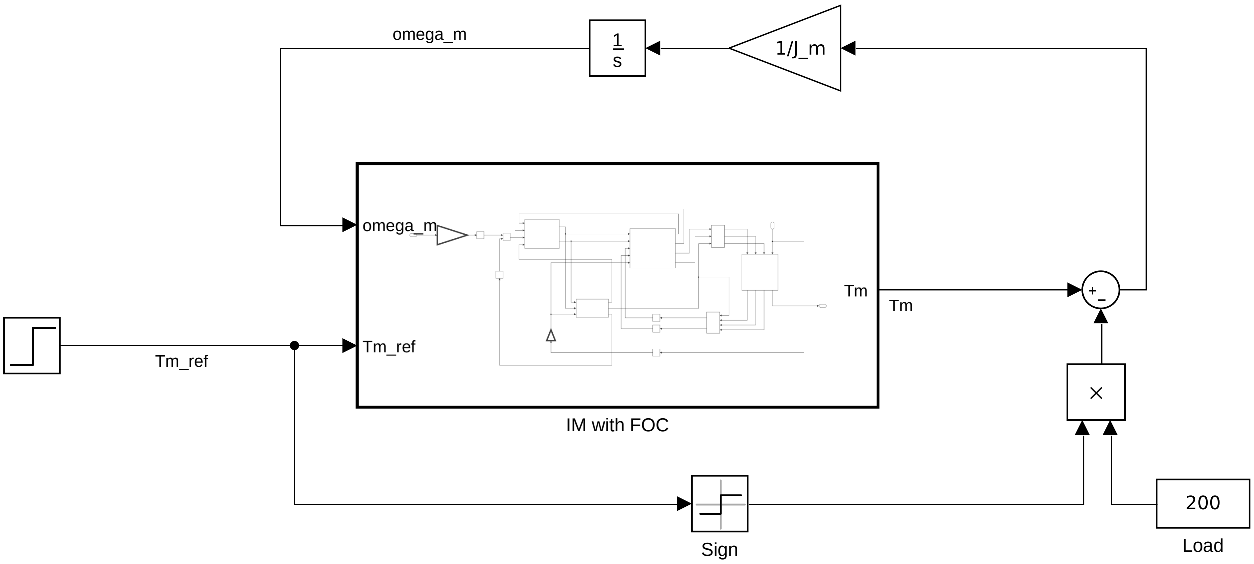


Figure E.2: Field-Weakening and Current Saturation setup.

E.10 Parameters

```
%% MAS409/MAS411 Main Project, group 8
%% Active heave compensation parameters

%% General constants
g = 9.81; % gravitational acceleration
rho = 1027; % density of sea water (approx) [kg/m^3]

%% Payload constants
m_pl = 12000; % mass payload [kg]
n_sh = 5; % number of sheaves []
n_msh = 2; % number of moving sheaves []
m_tb = 200+200*n_msh; % mass travelling block [kg]
n_fmsh = 2*n_msh; % pulley ratio [] seems to be correct: ...
    https:%www.translatorscafe.com/unit-converter/en-US/calculator/pulley-mechanical-adv
A_pl = 1.5; % projected area of payload [m^2]
V_pl = 2; % volume of payload [m^3], TODO(martin): temporarily set to zero ...
    to avoid bouancy
Cd = 1.8; % drag coefficient []
pl_pos_init = 7.5; % start position of payload ref seabed

%% Drum constants
J_d = 1; % drum inertia [kg*m^2]
d_d = 23.23e-2; % drum diameter [m]
r_d = d_d/2; % drum radius [m]
mu_d = 0.001; % viscous friction [n*s/m]

%% Seabed constants
k_sb = 1.8e6;
b_sb = 6.5e5;
p_pos_init = 350;   % initial position of platform ref seabed

%% Params from catalogue of chosen motor and gear
V_N = 400; % [V]
Pm_N_out = 132e3; % nominal output power [W]
fs_N = 50; % nominal stator frequency [Hz]
p = 4; % Nr. of poles []
n_p = p/2; % Nr. of pole pairs []
nm_N = 1487; % Speed [rpm]
PF_m = 0.86; % Power factor
Im_N = 232; % Nominal current [A]
Te_N = 847; % nominal torque [Nm]
J_m = 2.6; % Inertia [kgm2]
i_g = 4.5; % gear ratio [-]

J_eq = J_m*(i_g*n_fmsh)^2 + J_d*n_fmsh^2; % equivalent inertia at drum

Sm_N_in = 3*V_N*Im_N; % apparent input power [VA]
Pm_N_in = Sm_N_in*PF_m; % active input power [W]
eta_m_N = Pm_N_out/Pm_N_in; % nominal efficiency [-]
omega_e_N = 2*pi*fs_N;  % angular stator fequency [rad/s]
omega_m_N = nm_N*2*pi/60;   % nominal mechanical rotor speed [rad/s]
omega_r_N = n_p*omega_m_N;  % electrical rotor speed [rad/s]
s_N = (omega_e_N-omega_r_N)/omega_e_N; % slip [-]
omega_s = s_N*omega_e_N;    % electrical slip frequency [rad/s]

%% IM model params

R_s = 0.03211;
L_sl = 0.0001948;
```

```
R_r = 0.01002;
L_rl = 0.0004578;
L_m = 0.007917;

R_R = R_r*L_m^2/(L_rl+L_m)^2;
L_r = L_rl+L_m;
L_sigma = (L_m*(L_sl+L_rl)+L_sl*L_rl)/(L_m+L_rl);

psi_ref = L_m/sqrt(1+omega_s^2*((L_m+L_rl)^2)/R_r^2)*Im_N*3/2*sqrt(2);
I_max = 350*sqrt(2)*3/2;
I_nom = 232*sqrt(2)*3/2;
V_base = V_N*sqrt(2)/sqrt(3)*3/2;

L_sigma_hat = L_sigma;
R_R_hat = R_R;
L_m_hat = L_m;
L_r_hat = L_r;

t_r = 9e-3; % rise time set to 9ms for real time performance
w_BW = log(9)/t_r;

%Choosing an active resistance Ra such that the open loop system crossover ...
    frequency is
%equal to the bandwidth (closed loop system) results in:
R_a = w_BW*L_sigma_hat-R_R_hat;

% Following the direct synthesis method, using zero-pole cancellation, the ...
    PI gains are:
% Lecture 8, page 19
k_p = w_BW*L_sigma_hat;
k_i = w_BW^2*L_sigma_hat;

v_max = V_N*(sqrt(2)/sqrt(3))*(3/2); % Max amplitude of space vector total ...
    voltage. Lecture 8, page 3.



%% Params from cataloge of chosen drive
Pdrive_N = 132e3; % nominal power [W]
Idrive_N = 246; % nominal current [A]
Idrive_max = 350;    % max current [A]


%% Transformation matrices:
T_32 = [1 -1/2 -1/2; 0 sqrt(3)/2 -sqrt(3)/2 ];       % abc --> alfa-beta
T_23 = [2/3 0; -1/3 1/sqrt(3); -1/3 -1/sqrt(3)];     % alfa-beta --> abc
```

F PLC

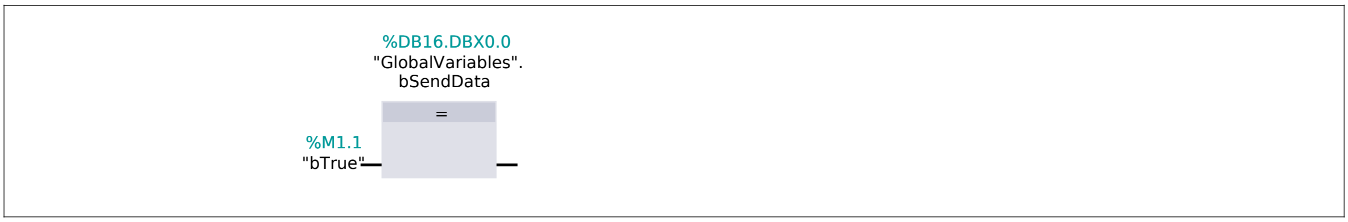
F.1 Main Programs & Function Blocks

CYC_INT [OB35]

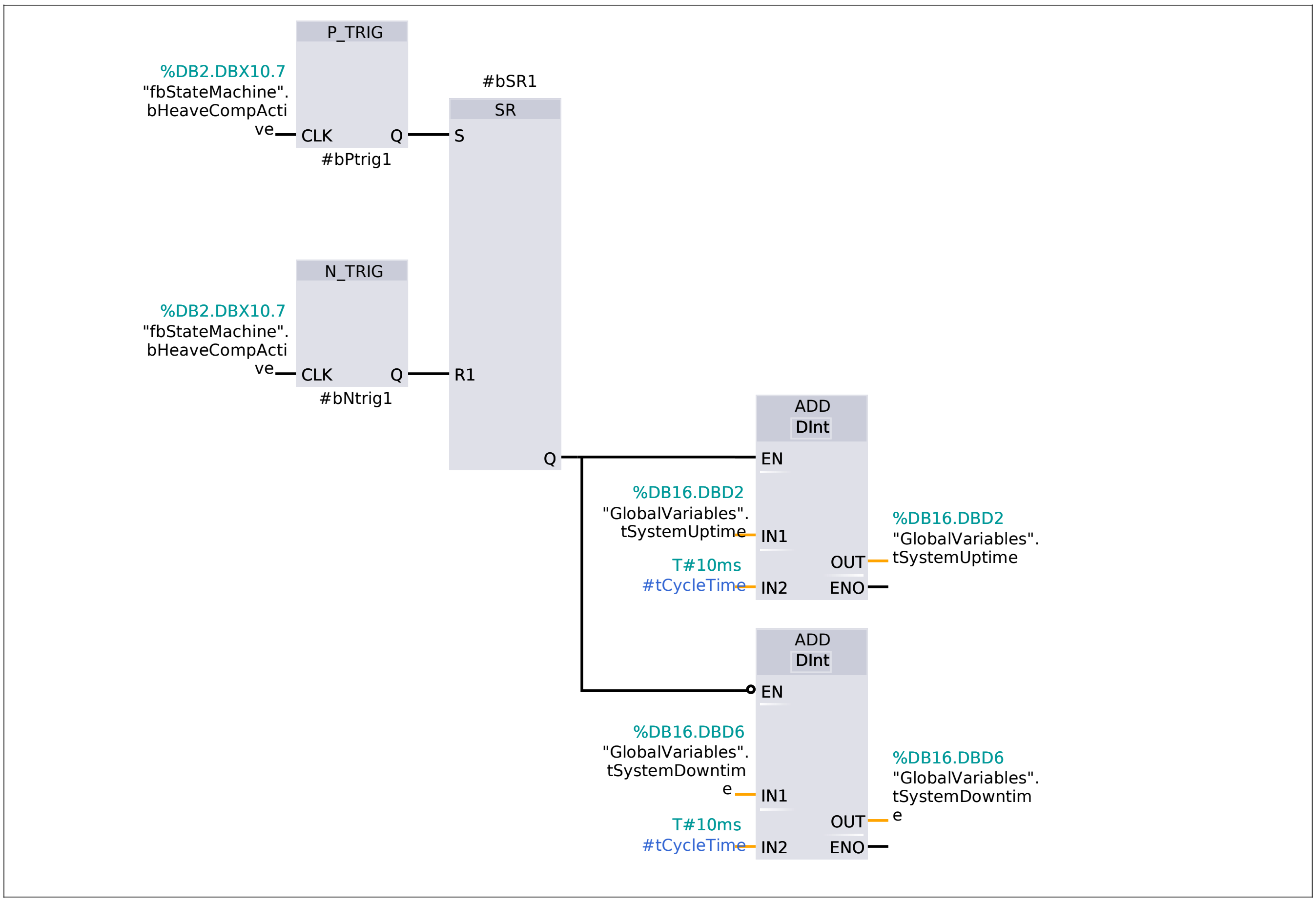
CYC_INT Properties					
General					
Name	CYC_INT	Number	35	Type	OB
Language	FBD	Numbering	Manual		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Default value	Super-vision	Comment
▼ Temp					
OB35_EV_CLASS	Byte	0.0			
OB35_STRT_INF	Byte	1.0			
OB35_PRIORITY	Byte	2.0			
OB35_OB_NUMBR	Byte	3.0			
OB35_RESERVED_1	Byte	4.0			
OB35_RESERVED_2	Byte	5.0			
OB35_PHASE_OFFSET	Word	6.0			
OB35_RESERVED_3	Int	8.0			
OB35_EXC_FREQ	Int	10.0			
OB35_DATE_TIME	Date_And_Time	12.0			
bPtrig1	Bool	20.0			
bNtrig1	Bool	20.1			
bSR1	Bool	20.2			
bPtrig2	Bool	20.3			
bNtrig2	Bool	20.4			
bSR2	Bool	20.5			
rSystemOnTime	Real	22.0			
rCurrentEnergy	Real	26.0			
rVelocityRefFromPositionController	Real	30.0			
rTmp	Real	34.0			
▼ Constant					
tCycleTime	Time		T#10ms		
rCycletime	Real		0.01		

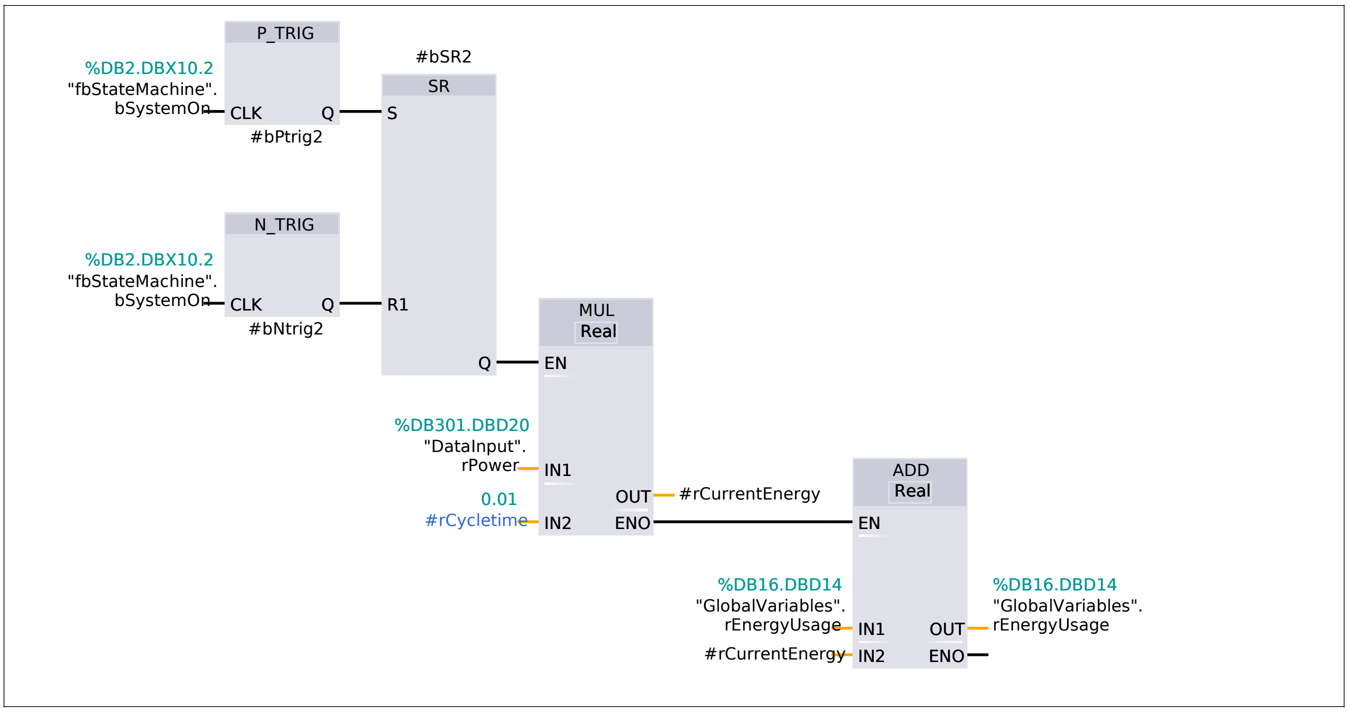
Network 1: Sendt data flag. Is set high every 10ms, and low right after.



Network 2: Calculate system uptime and downtime



Network 3: Calculate system on time



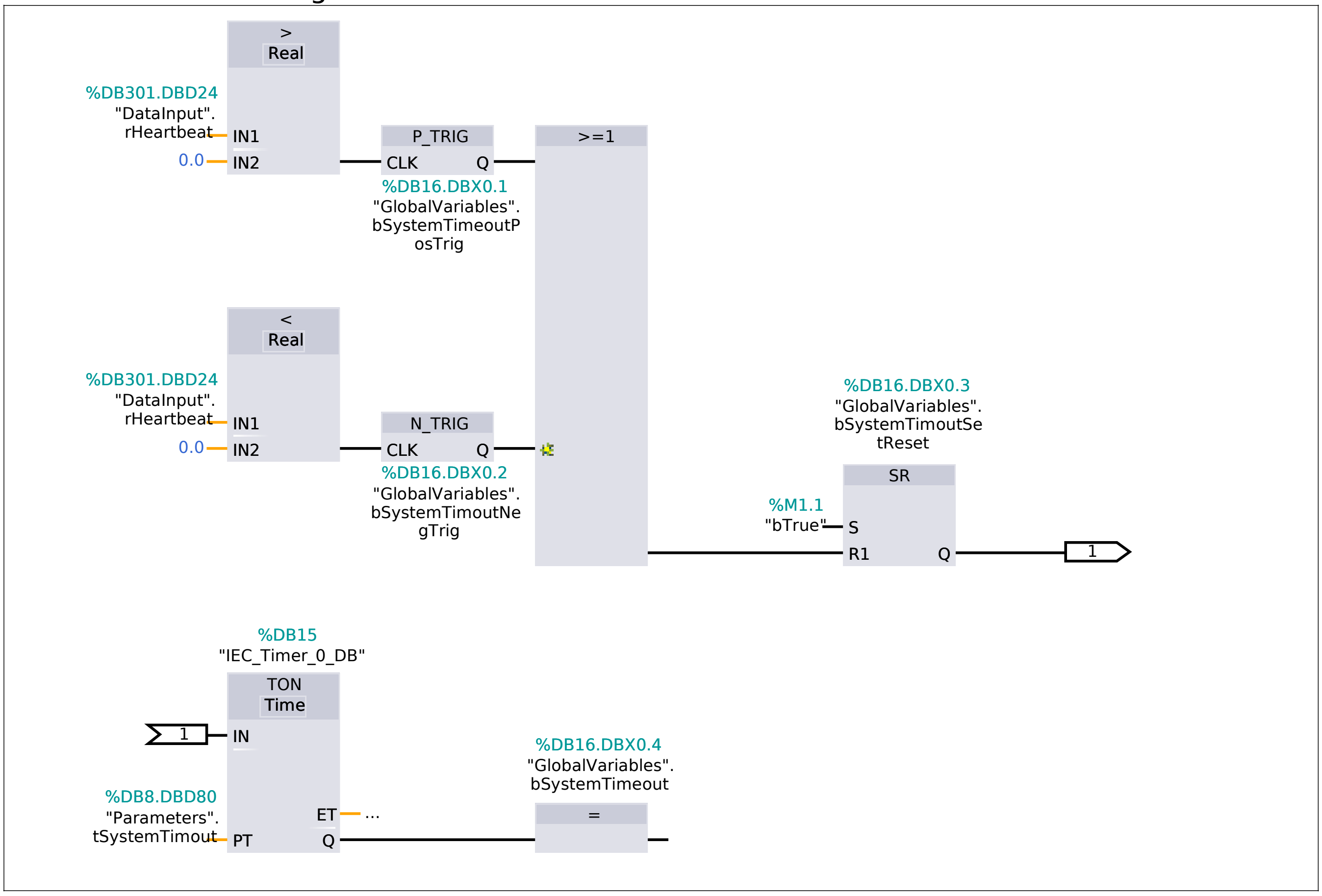
MAIN [OB1]

MAIN Properties					
General					
Name	MAIN	Number	1	Type	OB
Language	FBD	Numbering	Manual		
Information					
Title		Author		Comment	
Family		Version	1.0	User-defined ID	

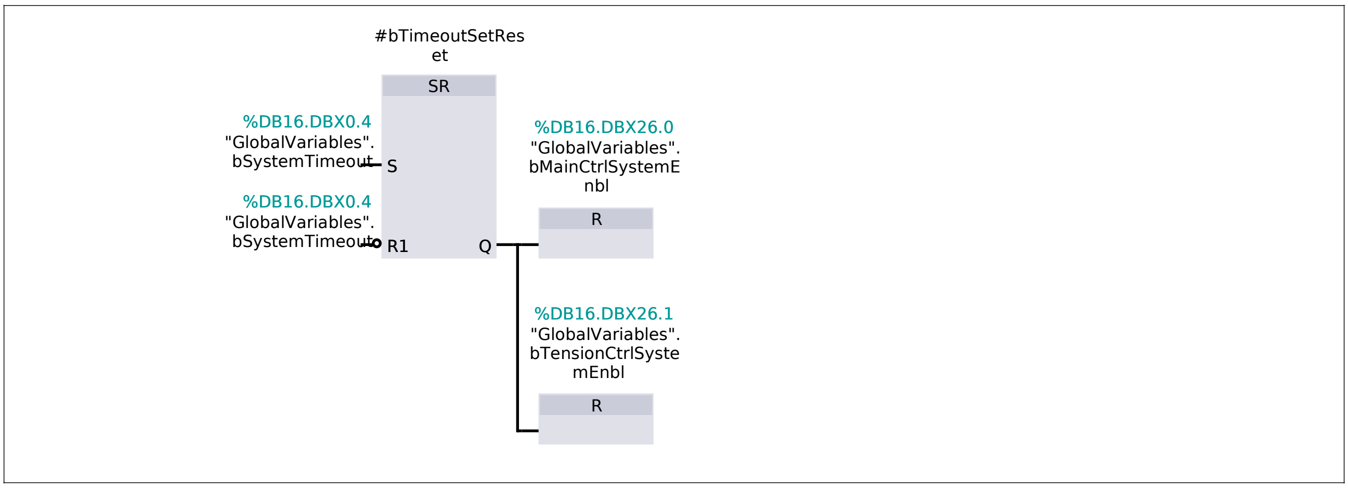
Name	Data type	Offset	Default value	Super-vision	Comment
Temp					
OB1_EV_CLASS	Byte	0.0			
OB1_SCAN_1	Byte	1.0			
OB1_PRIORITY	Byte	2.0			
OB1_OB_NUMBR	Byte	3.0			
OB1_RESERVED_1	Byte	4.0			
OB1_RESERVED_2	Byte	5.0			
OB1_PREV_CYCLE	Int	6.0			
OB1_MIN_CYCLE	Int	8.0			
OB1_MAX_CYCLE	Int	10.0			
OB1_DATE_TIME	Date_And_Time	12.0			
bStartTimer	Bool	20.0			
rVelocityRefFromPositionController	Real	22.0			
rIntegralContributionPos	Real	26.0			
rIntegralContributionVel	Real	30.0			
bHoldIntegralPos	Bool	34.0			
bHoldIntegralVel	Bool	34.1			
rTensionFeedback	Real	36.0			
rTmp	Real	40.0			
bTimeoutSetReset	Bool	44.0			
Constant					

Network 1: Check target heartbeat

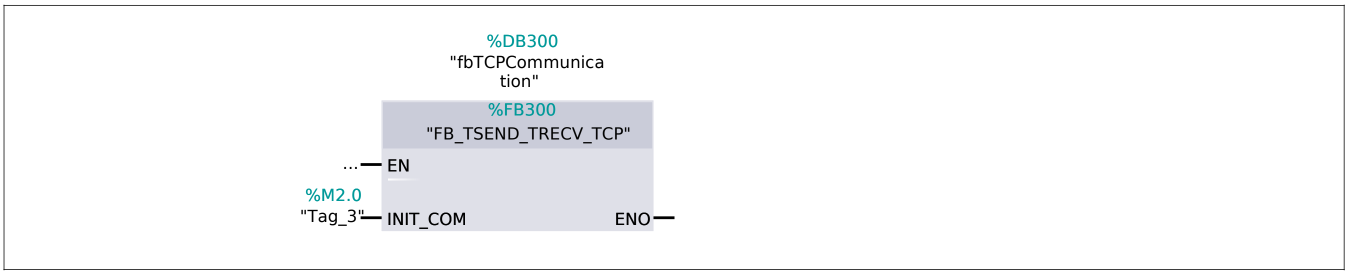
Network 1: Check target heartbeat



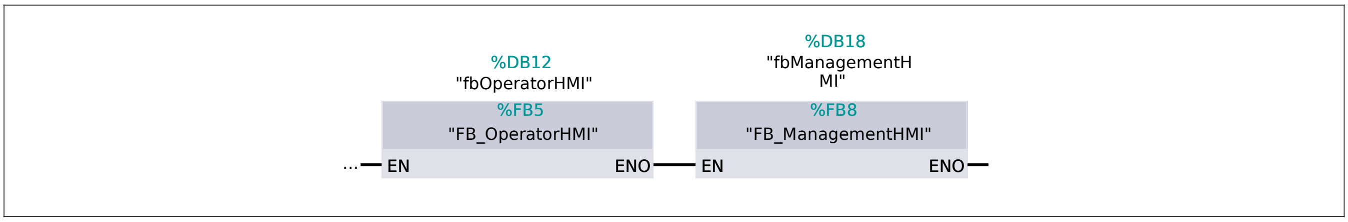
Network 2: Disable control systems when system timeout



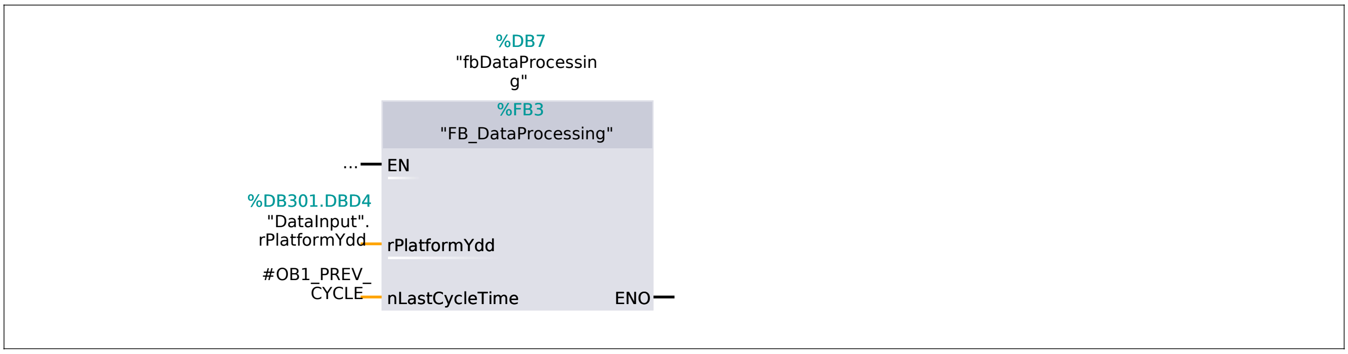
Network 3: fbTCPCommunication



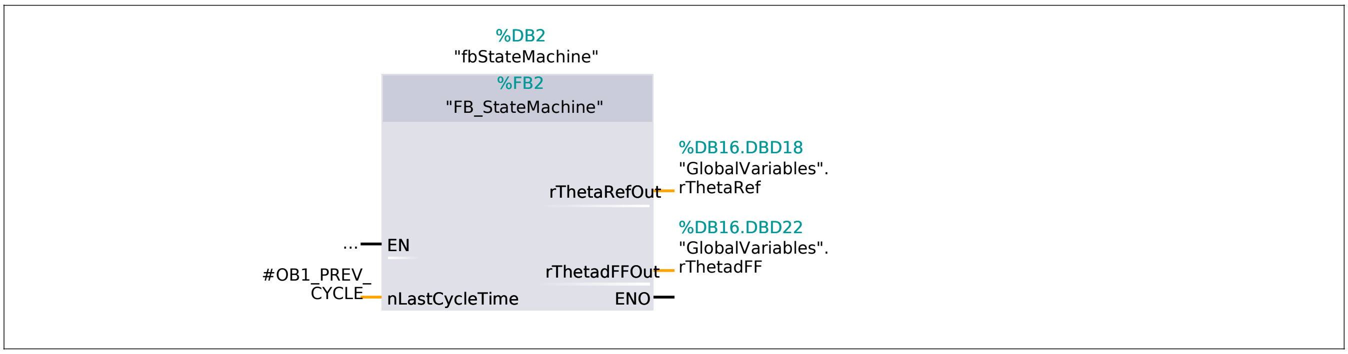
Network 4: fbOperatorHMI and fbManagementHMI



Network 5: fbControllerData



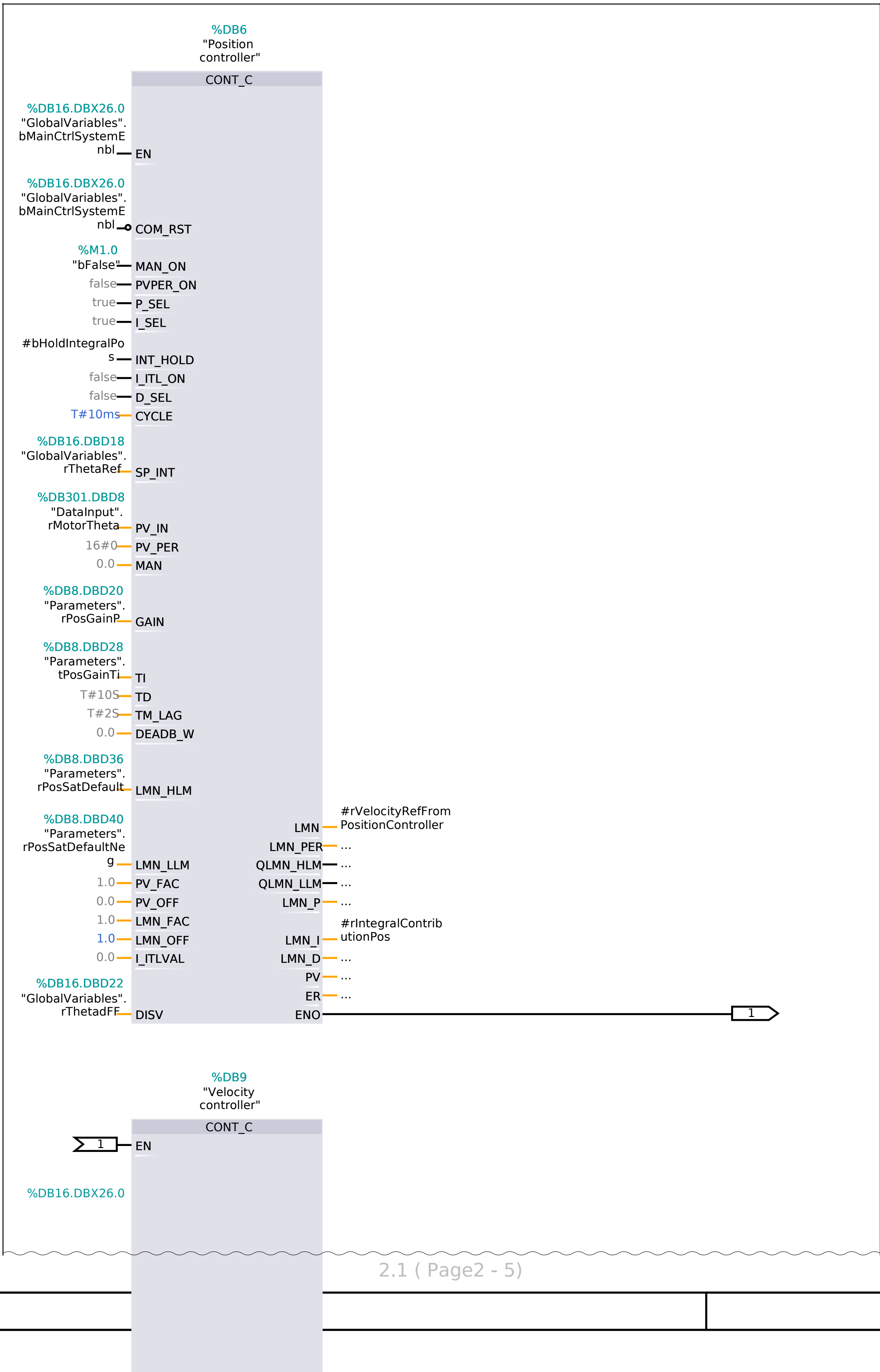
Network 6: fbSystemController



Network 7: Main control system

Control system is disabled if system timeout is true.

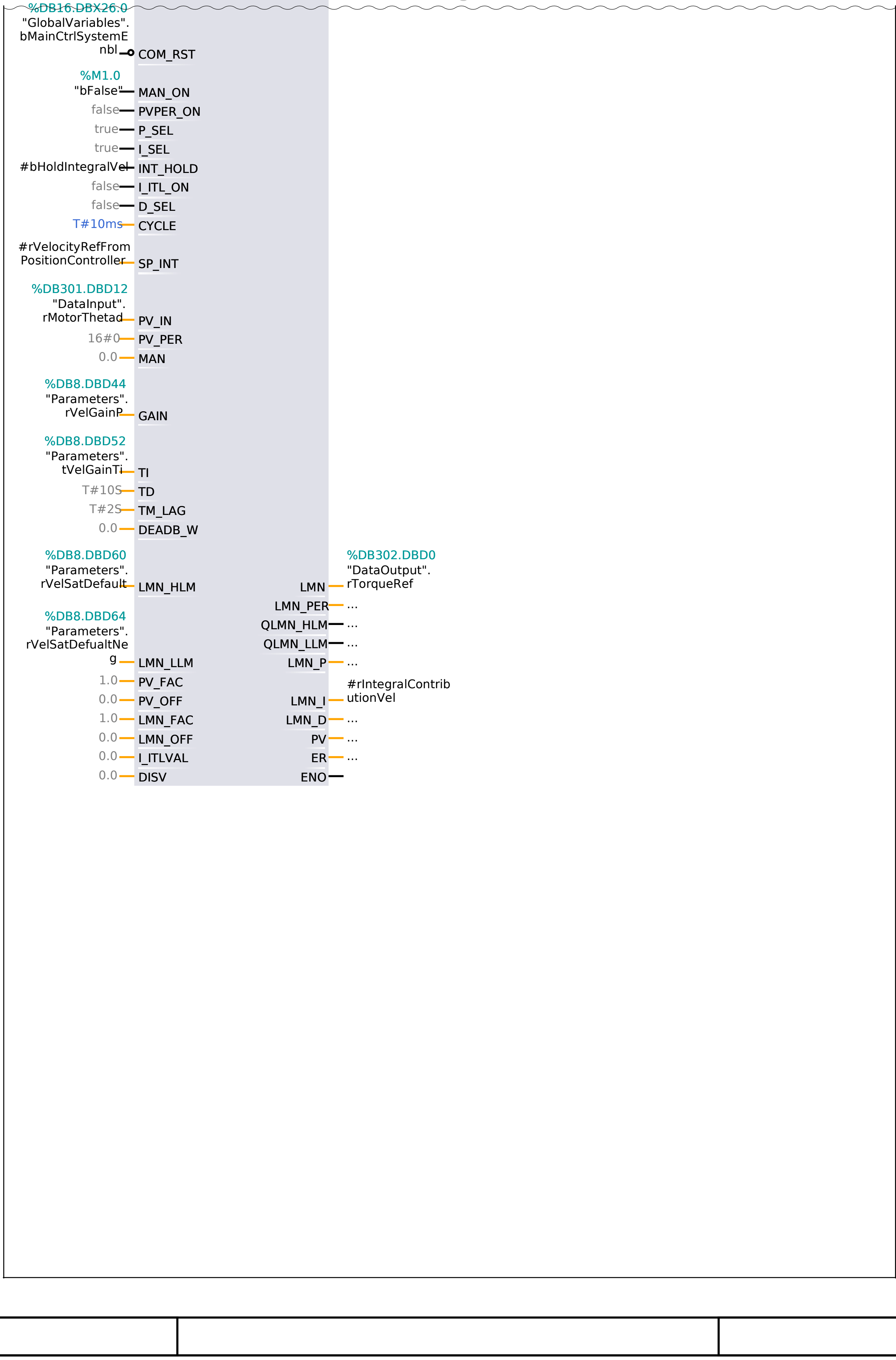
Network 7: Main control system (1.1 / 2.1)



2.1 (Page2 - 5)

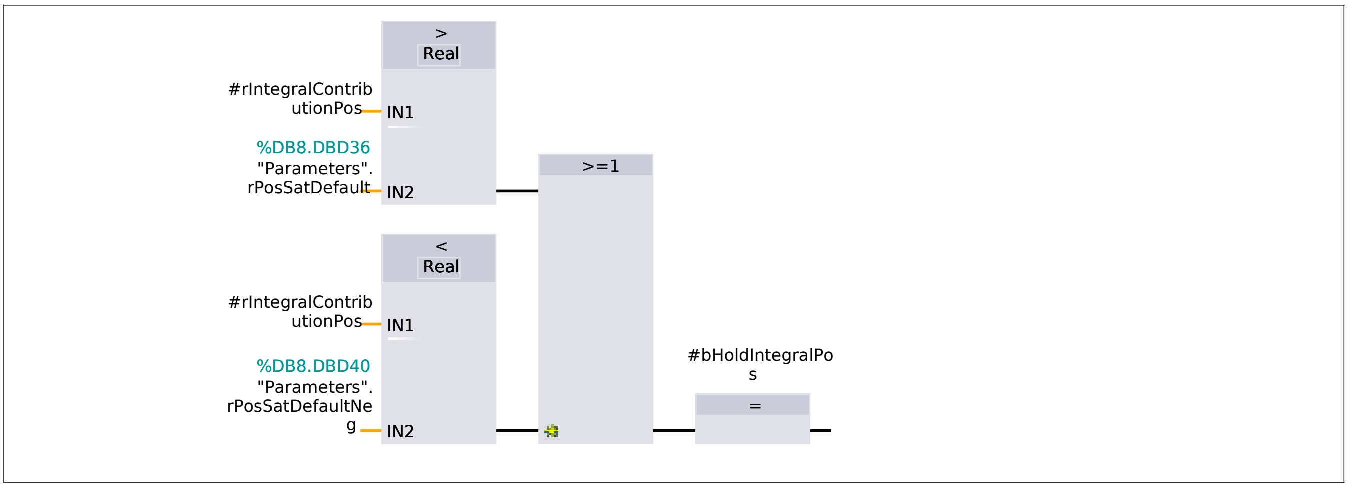
Network 7: Main (2.1)

1.1 (Page2 - 4)



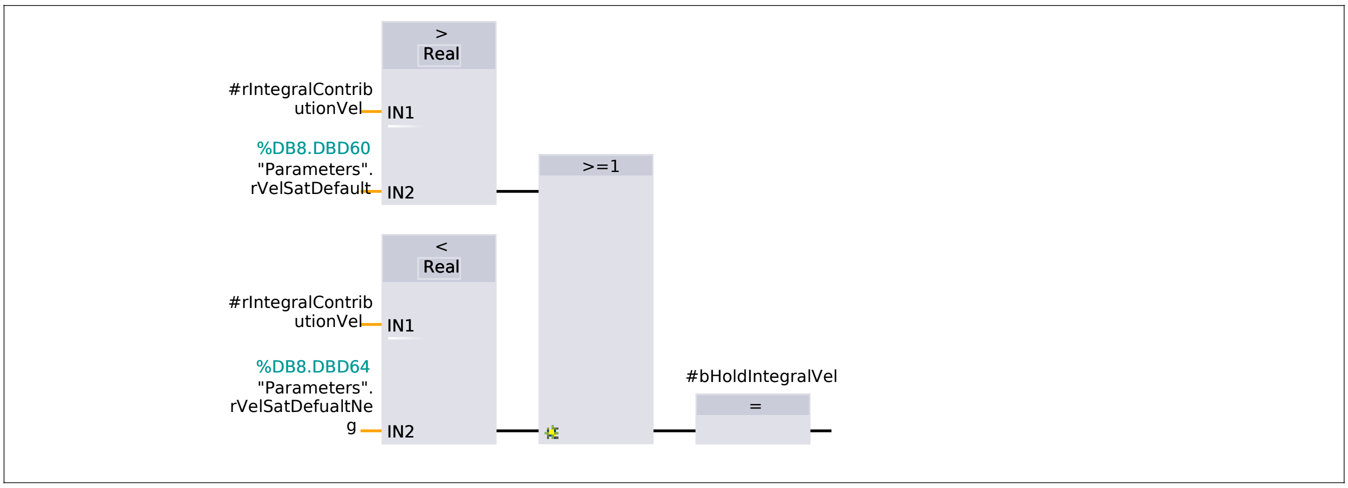
Network 8: Anti-windup position

Saturate integrator term of velocity controller if the contribution is larger than the set limits.

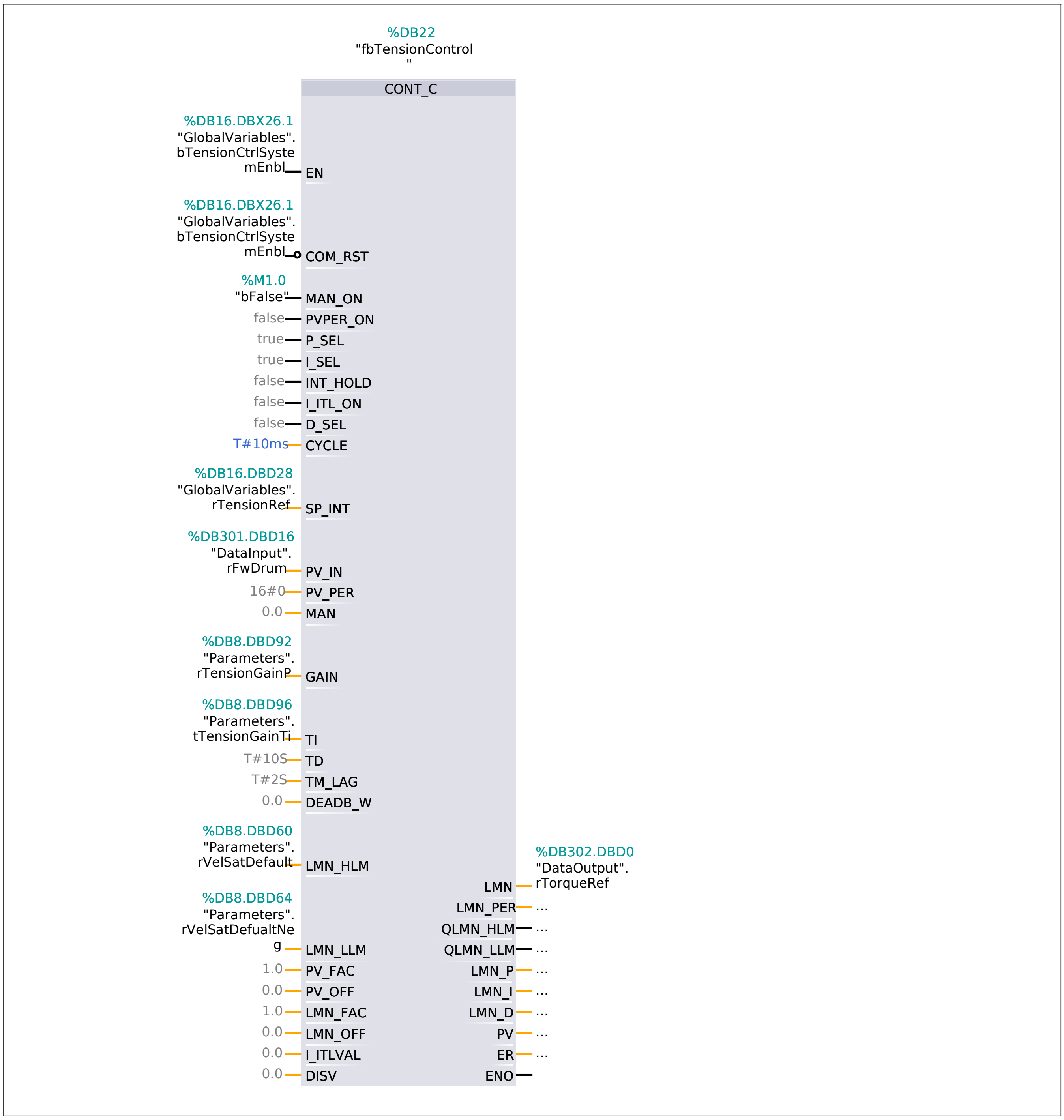


Network 9: Anti-windup velocity

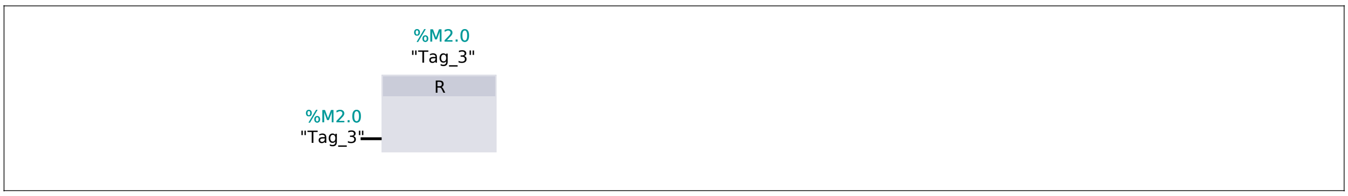
Saturate integrator term of velocity controller if the contribution is larger than the set limits.



Network 10: Tension control system



Network 11:



Network 12:

```
0001    CLR
```

```
      =     "bFalse"
      SET
      =     "bTrue"
```

START_UP [OB100]

START_UP Properties					
General					
Name	START_UP	Number	100	Type	OB
Language	STL	Numbering	Manual		
Information					
Title		Author		Comment	
Family		Version	1.4	User-defined ID	

Name	Data type	Offset	Default value	Super-vision	Comment
▼ Temp					
▼ Default	Array[1..20] of Byte	0.0			
Default[1]	Byte	0.0			
Default[2]	Byte	1.0			
Default[3]	Byte	2.0			
Default[4]	Byte	3.0			
Default[5]	Byte	4.0			
Default[6]	Byte	5.0			
Default[7]	Byte	6.0			
Default[8]	Byte	7.0			
Default[9]	Byte	8.0			
Default[10]	Byte	9.0			
Default[11]	Byte	10.0			
Default[12]	Byte	11.0			
Default[13]	Byte	12.0			
Default[14]	Byte	13.0			
Default[15]	Byte	14.0			
Default[16]	Byte	15.0			
Default[17]	Byte	16.0			
Default[18]	Byte	17.0			
Default[19]	Byte	18.0			
Default[20]	Byte	19.0			
Constant					

Network 1:

```
0001    ON   "Always 1"
0002    O    "Always 1"
0003    =    "Always 1"
```

Network 2:

```
0001    AN   "Always 0"
0002    A    "Always 0"
0003    =    "Always 0"
```

Network 3:

```
ON    "Always 1"
O     "Always 1"
=     "Tag_3"
```

Network 4:

```
CLR
=     "fbTCPCommunication".C1.CONNECTED
```

FB_DataProcessing [FB3]

FB_DataProcessing Properties					
General					
Name	FB_DataProcessing	Number	3	Type	FB
Language	SCL	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input									
rPlatformYdd	Real	0.0	0.0	True	True	True	False		
nLastCycleTime	Int	4.0	0	True	True	True	False		
Output									
InOut									
▼ Static									
bFirstRunDone	Bool	6.0	false	True	True	True	False		
rK	Real	8.0	0.0	True	True	True	False		
rMotorThetad_pl	Real	12.0	0.0	True	True	True	False		
rMotorThetad_p	Real	16.0	0.0	True	True	True	False		
rPlatformYd	Real	20.0	0.0	True	True	True	False		
rPlatformYdFiltered	Real	24.0	0.0	True	True	True	False		
rLastycleTime	Real	28.0	0.0	True	True	True	False		
Temp									
Constant									

```
// Cycle time in seconds
#rLastycleTime := INT_TO_REAL(#nLastCycleTime) * 10 ** (-3);

// Calculate K constant, world -> motor ratio
IF NOT #bFirstRunDone THEN
  #rK := ("Parameters".rSheaveRatio * "Parameters".rGearRatio) / "Parameters".rDrumRadius;
  #bFirstRunDone := TRUE;
END_IF;

// Integrate platform acc to get platform vel
IF NOT "GlobalVariables".bSystemTimeout THEN
  #rPlatformYd := #rPlatformYd + #rPlatformYdd * #rLastycleTime;
```

```

// Filter platform vel
"fbLowPassFilterPlatformYd"(rCutoffFrequency := 0.5305, // break frequency tuned in matlab
             rDt := #rLastycleTime,
             rX := #rPlatformYd,
             rY => #rPlatformYdFiltered);
//#rPlatformYdFiltered := 0.0;
ELSE
  #rPlatformYd := 0.0;
  #rPlatformYdFiltered := 0.0;
END_IF;
```

Symbol	Address	Type	Comment
"GlobalVariables".bSystemTimeout	%DB16.DBX0.4	Bool	
"Parameters".rDrumRadius	%DB8.DBD8	Real	Drum radius.
"Parameters".rGearRatio	%DB8.DBD0	Real	System gear ratio.
"Parameters".rSheaveRatio	%DB8.DBD4	Real	System sheave ratio.
#bFirstRunDone		Bool	
#nLastCycleTime		Int	
#rK		Real	
#rLastycleTime		Real	
#rPlatformYd		Real	
#rPlatformYdd		Real	
#rPlatformYdFiltered		Real	

FB_ManagementHMI [FB8]

FB_ManagementHMI Properties					
General					
Name	FB_ManagementHMI	Number	8	Type	FB
Language	SCL	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
rPowerUsage	Real	0.0	0.0	True	True	True	False		
rEnergyUsage	Real	4.0	0.0	True	True	True	False		
rDowntime	Real	8.0	0.0	True	True	True	False		
rUptime	Real	12.0	0.0	True	True	True	False		
rPayloadY	Real	16.0	7.5	True	True	True	False		
rPlatformY	Real	20.0	350.0	True	True	True	False		
Temp									
Constant									

```
// Update variables
#rPowerUsage := "DataInput".rPower*(10**-3);
#rEnergyUsage := "GlobalVariables".rEnergyUsage*(10**-3);
#rDowntime := DINT_TO_REAL(TIME_TO_DINT("GlobalVariables".tSystemDowntime))/
    (1000.0*3600.0);
#rUptime := DINT_TO_REAL(TIME_TO_DINT("GlobalVariables".tSystemUptime)) /
    (1000.0 * 3600.0);
#rPayloadY := "DataInput".rPayloadY;
#rPlatformY := ABS("DataInput".rMotorTheta / "fbDataProcessing".rK) + #rPay-
    loadY;
```

Symbol	Address	Type	Comment
"DataInput".rMotorTheta	%DB301.DBD8	Real	
"DataInput".rPayloadY	%DB301.DBD0	Real	
"DataInput".rPower	%DB301.DBD20	Real	
"fbDataProcessing".rK	%DB7.DBD8	Real	
"GlobalVariables".rEnergyUsage	%DB16.DBD14	Real	

Symbol	Address	Type	Comment
"GlobalVariables".tSystemDowntime	%DB16.DBD6	Time	
"GlobalVariables".tSystemUptime	%DB16.DBD2	Time	
#rDowntime		Real	
#rEnergyUsage		Real	
#rPayloadY		Real	
#rPlatformY		Real	
#rPowerUsage		Real	
#rUptime		Real	

FB_OperatorHMI [FB5]

FB_OperatorHMI Properties					
General					
Name	FB_OperatorHMI	Number	5	Type	FB
Language	SCL	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
Input									
Output									
InOut									
▼ Static									
bFirstRunDone	Bool	0.0	FALSE	True	True	True	False		Initializing flag.
bBtnOn	Bool	0.1	FALSE	True	True	True	False		Button on.
bBtnOnEnbl	Bool	0.2	TRUE	True	True	True	False		
bBtnManual	Bool	0.3	FALSE	True	True	True	False		Button manual.
bManualBtnEnbl	Bool	0.4	FALSE	True	True	True	False		Manual button enabled if true.
bBtnAutomatic	Bool	0.5	FALSE	True	True	True	False		Button automatic.
bAutoButtonsEnbl	Bool	0.6	FALSE	True	True	True	False		Auto buttons are enabled if true.
bBtnEmg	Bool	0.7	FALSE	True	True	True	False		Button emergency.
bIdle	Bool	1.0	FALSE	True	True	True	False		Idle state.
bShowSetpointAndError	Bool	1.1	FALSE	True	True	True	False		True if setpoint and error is visible.
bBtnService	Bool	1.2	FALSE	True	True	True	False		Button service.
bServiceBtnEnbl	Bool	1.3	FALSE	True	True	True	False		
rPositionGainP	Real	2.0	1.0	True	True	True	False		
rPositionGainT	Real	6.0	0.0	True	True	True	False		
rVelocityGainP	Real	10.0	1.0	True	True	True	False		
rVelocityGainT	Real	14.0	1.0	True	True	True	False		
bServiceSetValues	Bool	18.0	FALSE	True	True	True	False		

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
bServiceSetDefault	Bool	18.1	TRUE	True	True	True	False		
bSliderEnblSwitch	Bool	18.2	FALSE	True	True	True	False		Switch that enables or disables position slider ref.
bManualAndHeaveEnbl	Bool	18.3	FALSE	True	True	True	False		True if in manual mode and ahc is enabled.
bExtEnblSwitch	Bool	18.4	FALSE	True	True	True	False		Switch that enables or disables external position ref.
bManualPosRef	Bool	18.5	FALSE	True	True	True	False		Manual position state in manual state (confusing? u stupid).
bHeaveCompActive	Bool	18.6	FALSE	True	True	True	False		Heave compensation active flag.
bHeaveBtnEnbl	Bool	18.7	FALSE	True	True	True	False		True if heave comp button is active.
nAnimPos	Int	20.0	0	True	True	True	False		Payload animation position as integer [0-100].
rPlatformY	Real	22.0	350.0	True	True	True	False		Platform position in global frame.
rPlatformY_moh	Real	26.0	0.0	True	True	True	False		Platform position in reference to sea level.
rPlatformYd	Real	30.0	0.0	True	True	True	False		Platform velocity.
rPayloadY	Real	34.0	7.5	True	True	True	False		Payload position in global frame.
rPayloadYLast	Real	38.0	7.5	True	True	True	False		
rPayloadYRef	Real	42.0	7.5	True	True	True	False		Payload position reference in global frame.
rPayloadYError	Real	46.0	0.0	True	True	True	False		Payload position error.
rPayloadYd	Real	50.0	0.0	True	True	True	False		Payload velocity in global frame.
rPayloadYdFiltered	Real	54.0	0.0	True	True	True	False		
rMotorPowerOut	Real	58.0	0.0	True	True	True	False		Motor power output.
nPayloadYRef	Int	62.0	7	True	True	True	False		Manual mode payload position reference.
rPayloadYdRef	Real	64.0	7.5	True	True	True	False		Manual mode payload velocity reference.
nManualJogDir	Int	68.0	0	True	True	True	False		1 if up is pressed, -1 if down is pressed.
rMotorRPM	Real	70.0	0.0	True	True	True	False		Motor speed [rpm].

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
rDrumRPM	Real	74.0	0.0	True	True	True	False		Drum speed [rpm].
rTheta	Real	78.0	0.0	True	True	True	False		
rThetaRef	Real	82.0	0.0	True	True	True	False		
rThetaError	Real	86.0	0.0	True	True	True	False		
rWireForce	Real	90.0	0.0	True	True	True	False		Wire force at drum [kN].
rSeabedForce	Real	94.0	0.0	True	True	True	False		
rHookLoad	Real	98.0	0.0	True	True	True	False		Hook load [kg].
bBtnGenerateTrajectory	Bool	102.0	FALSE	True	True	True	False		Button to generate trajectory.
bTrajectoryGenerated	Bool	102.1	TRUE	True	True	True	False		True if trajectory is generated.
bRunTrajectoryBtnEnbl	Bool	102.2	FALSE	True	True	True	False		Run trajectory button is enabled if true.
bTrajectory75Btn	Bool	102.3	FALSE	True	True	True	False		Predefined trajectory params to 7.5 button.
bTrajectory5Btn	Bool	102.4	FALSE	True	True	True	False		Predefined trajectory params to 5 button.
bTrajectoryLandBtn	Bool	102.5	FALSE	True	True	True	False		Predefined landing sequence button.
nLandingSequence	Int	104.0	0	True	True	True	False		Initiate landing sequence.
rTrajectoryVelViz	Real	106.0	0.0	True	True	True	False		
rTrajectoryAccVizUpper	Real	110.0	0.0	True	True	True	False		
rTrajectoryAccVizLower	Real	114.0	0.0	True	True	True	False		
bTrajectoryCompleted	Bool	118.0	FALSE	True	True	True	False		
bTrajectoryCanceled	Bool	118.1	FALSE	True	True	True	False		
bTrajectoryVisFalling	Bool	118.2	TRUE	True	True	True	False		True if trajectory is negative (endpoint lower than current).
rTrajectoryYEnd	Real	120.0	1.0	True	True	True	False		Trajectory endpoint.
rTrajectoryTp	Real	124.0	5.0	True	True	True	False		Time to complete trajectory.
rTrajectoryTime	Real	128.0	10.0	True	True	True	False		
rK	Real	132.0	1.0	True	False	False	False		Conversion factor global to motor.

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
nCurrentState	Int	136.0	0	True	True	True	False		System current state, from fbSystemController.
nRequestedState	Int	138.0	0	True	True	True	False		System requested state.
nLastState	Int	140.0	0	True	True	True	False		System state last run.
sInfoMsg	String	142.0	''	True	True	True	False		Info string, shows up in top right corner of hmi.
bTimerStart	Bool	398.0	TRUE	True	True	True	False		
bTimerDone	Bool	398.1	false	True	True	True	False		
tTimerET	Time	400.0	T#0ms	True	True	True	False		
rTestVariable	Real	404.0	0.0	True	True	True	False		
nTrendManualSelection	Int	408.0	0	True	True	True	False		
rTrendManualData	Real	410.0	0.0	True	True	True	False		
rTorqueRef	Real	414.0	0.0	True	True	True	False		
▼ Temp									
rTrajectoryTimeLowerLimit	Real	0.0							
Constant									

```
// This function block contains all hmi variables and
// connects them to the logic of the system.

// Initialize variables
IF NOT #bFirstRunDone THEN
  #rK := ("Parameters".rSheaveRatio * "Parameters".rGearRatio) / "Parameters".rDrumRadius;
  #rPositionGainP := "Parameters".rPosGainPDefault;
  #rVelocityGainP := "Parameters".rVelGainP;
  #rVelocityGainT := DINT_TO_REAL(TIME_TO_DINT("Parameters".tVelGainTiDefault));
  #bFirstRunDone := TRUE;
END_IF;

// -----------------
// --- Update hmi ---
// -----------------

// Derivate payload position
"fbDerivativeTimer".TON(IN := #bTimerStart,
           PT := T#100ms,
           Q => #bTimerDone,
           ET => #tTimerET);
```

```
IF NOT #bTimerStart THEN
  #bTimerStart := TRUE;
END_IF;

IF #bTimerDone THEN
  #rPayloadYd := (#rPayloadY - #rPayloadYLast) / 0.1;
  "fbLowPassFilterPayloadYd"(rCutoffFrequency := 0.1, // break frequency found in matlab
              rDt := 0.1,
              rX := #rPayloadYd,
              rY => #rPayloadYdFiltered);
  #rPayloadYLast := #rPayloadY;
  #bTimerStart := FALSE;
END_IF;

// Update hmi instrument variables
#rPayloadY := "DataInput".rPayloadY;
#rPayloadYRef := "fbStateMachine".rPayloadYRef;
#rPayloadYError := (#rPayloadYRef - #rPayloadY)*(10**2); // [cm]
#rPayloadYdRef := "fbStateMachine".rPayloadYdRef;
#rPlatformY := #rPayloadY - "DataInput".rMotorTheta / #rK;
#rPlatformY_moh := #rPlatformY - 350.0;
#rPlatformYd := #rPayloadYd - "DataInput".rMotorThetad / #rK;
#rTrajectoryTime := "fbStateMachine".rTrajectoryTime;
#rMotorRPM := ABS("DataInput".rMotorThetad * (60 / (2 * "Parameters".#rPi)));
#rDrumRPM := ("DataInput".rMotorThetad * (60 / (2 * "Parameters".#rPi))) / "Parameters".rGearRatio;
#rTheta := "DataInput".rMotorTheta;
#rThetaRef := "GlobalVariables".rThetaRef;
#rThetaError := #rThetaRef - #rTheta;
#rWireForce := ABS("DataInput".rFwDrum) * 10**(-3);
#rHookLoad := (#rWireForce * "Parameters".rSheaveRatio / 9.81);
#rMotorPowerOut := ABS("DataInput".rMotorThetad) * #rWireForce * "Parameters".rDrumRadius / "Parameters".rGearRatio;
#rTorqueRef := "DataOutput".rTorqueRef;

// Seabed force
IF #rPayloadY < 0.0 THEN
  #rSeabedForce := ((("Parameters".rPayloadMass + "Parameters".rTravelingBlockMass)*9.81)*(10**-3)) - ABS(#rWireForce * "Parameters".rSheaveRatio);
ELSE
  #rSeabedForce := 0.0;
END_IF;

// Trend manual selection
CASE #nTrendManualSelection OF
  1: // Drum torque
    #rTrendManualData := #rWireForce * "Parameters".rDrumRadius;
  2: // Wire force
    #rTrendManualData := #rWireForce;
  3: // Motor Power
    #rTrendManualData := #rMotorPowerOut;
  ELSE: // Drum RPM
    #rTrendManualData := #rDrumRPM;
END_CASE;

// Animate payload
#nAnimPos := REAL_TO_INT(#rPayloadY*6);

```

```
// -------------------------------------
// --- Update system controller flags ---
// -------------------------------------

// Get current state from system controller and update buttons accordingly
// If system controller is doing something (changing states etc.) the hmi
// should not overrule. Wait until system controller has done its thing,
// and update the hmi flag values from the system controller flag values.
// Else update the system controller flag values with the hmi flag values.
#nCurrentState := "fbStateMachine".nCurrentState;
#nRequestedState := "fbStateMachine".nRequestState;
IF (#nCurrentState <> #nLastState) OR (#nCurrentState <> #nRequestedState) THEN
  #bBtnEmg := "fbStateMachine".bSystemEmg;
  #bBtnOn := "fbStateMachine".bSystemOn;
  #bIdle := "fbStateMachine".bSystemIdle;
  #bBtnManual := "fbStateMachine".bSystemManual;
  #bBtnAutomatic := "fbStateMachine".bSystemAutomatic;
  #bBtnService := "fbStateMachine".bSystemService;
  #bHeaveCompActive := "fbStateMachine".bHeaveCompActive;
  #bManualPosRef := "fbStateMachine".bManualPosRef;
  #bTrajectoryCompleted := "fbStateMachine".bTrajectoryCompleted;
  #bTrajectoryCanceled := "fbStateMachine".bTrajectoryCanceled;
  #bTrajectoryLandBtn := "fbStateMachine".bTrajectoryLandBtn;
  #nLastState := #nCurrentState;
ELSE
  "fbStateMachine".bSystemEmg := #bBtnEmg;
  "fbStateMachine".bSystemOn := #bBtnOn;
  "fbStateMachine".bSystemIdle := #bIdle;
  "fbStateMachine".bSystemManual := #bBtnManual;
  "fbStateMachine".bSystemAutomatic := #bBtnAutomatic;
  "fbStateMachine".bSystemService := #bBtnService;
  "fbStateMachine".bHeaveCompActive := #bHeaveCompActive;
  "fbStateMachine".bManualPosRef := #bManualPosRef;
  "fbStateMachine".bTrajectoryCompleted := #bTrajectoryCompleted;
  "fbStateMachine".bTrajectoryCanceled := #bTrajectoryCanceled;
  "fbStateMachine".bTrajectoryLandBtn := #bTrajectoryLandBtn;
END_IF;

// Get info msg
#sInfoMsg := "fbStateMachine".sInfoMsg;

// --------------------------------------
// --- Activate and deactivate buttons ---
// --------------------------------------

CASE #nCurrentState OF
  0: // OFF
    #bHeaveBtnEnbl := FALSE;
    #bAutoButtonsEnbl := FALSE;
    #bRunTrajectoryBtnEnbl := FALSE;
    #bShowSetpointAndError := FALSE;
    #bServiceBtnEnbl := FALSE;
    #bManualBtnEnbl := FALSE;

    IF "GlobalVariables".bSystemTimeout OR #bBtnEmg THEN
      IF "GlobalVariables".bSystemTimeout THEN
        #sInfoMsg := 'System timeout';
      END_IF;
```

```
      #bBtnOnEnbl := FALSE;
    ELSE
      #sInfoMsg := 'System online';
      #bBtnOnEnbl := TRUE;
    END_IF;

  1: // IDLE
    #bHeaveBtnEnbl := TRUE;
    #bManualAndHeaveEnbl := FALSE;
    #bManualBtnEnbl := TRUE;

    IF #bHeaveCompActive THEN
      #bAutoButtonsEnbl := TRUE;
      #bShowSetpointAndError := TRUE;
      #bServiceBtnEnbl := TRUE;

      IF #bTrajectory75Btn THEN
        #rTrajectoryYEnd := 7.5;
        #rTrajectoryTp := 10.0;
        #bTrajectory75Btn := FALSE;
      ELSIF #bTrajectory5Btn THEN
        #rTrajectoryYEnd := 5.0;
        #rTrajectoryTp := 10.0;
        #bTrajectory5Btn := FALSE;
      END_IF;

      // Check landing sequence
      IF #bTrajectoryLandBtn AND #rTrajectoryYEnd <> 0.0 THEN
        #rTrajectoryYEnd := 0.0;
        #rTrajectoryTp := 15.0;
        #bTrajectoryGenerated := FALSE;
      END_IF;

      // Check trajectory endpoint and time limits
      IF ((#rTrajectoryYEnd <> "fbStateMachine".rTrajectoryYEnd) OR (#rTrajectoryTp <> "fbStateMachine".rTrajectoryTp)) AND NOT #bTrajectoryLandBtn THEN
        #bTrajectoryGenerated := FALSE;

        // Limit trajectory end position
        IF #rTrajectoryYEnd < 1.0 THEN
          #rTrajectoryYEnd := 1.0;
        ELSIF #rTrajectoryYEnd > 15.0 THEN
          #rTrajectoryYEnd := 15.0;
        END_IF;

        // Limit trajectory time
        #rTrajectoryTimeLowerLimit := ABS(#rTrajectoryYEnd-#rPayloadY)*1.5;
        IF #rTrajectoryTp < #rTrajectoryTimeLowerLimit THEN
          #rTrajectoryTp := #rTrajectoryTimeLowerLimit;
        END_IF;
      END_IF;

      // Generate trajectory
      IF #bBtnGenerateTrajectory AND NOT #bTrajectoryGenerated THEN
        // Check direction of path for hmi visualization
        IF (#rTrajectoryYEnd - #rPayloadY) > 0 THEN
          #bTrajectoryVisFalling := FALSE;
        ELSE
          #bTrajectoryVisFalling := TRUE;
```

```
      END_IF;

      #rTrajectoryVelViz := "F_QuinticMaxVel"(rTp := #rTrajectoryTp,
                          rY0 := #rPayloadY,
                          rY1 := #rTrajectoryYEnd);
      #rTrajectoryAccVizUpper := "F_QuinticMaxAcc"(rTp := #rTrajectoryTp,
                          rY0 := #rPayloadY,
                          rY1 := #rTrajectoryYEnd);
      #rTrajectoryAccVizLower := -#rTrajectoryAccVizUpper;
      "fbStateMachine".rTrajectoryYEnd := #rTrajectoryYEnd;
      "fbStateMachine".rTrajectoryTp := #rTrajectoryTp;
      #bBtnGenerateTrajectory := FALSE;
      #bTrajectoryGenerated := TRUE;
    END_IF;

    // Enable or disable run trajectory button
    IF #bTrajectoryGenerated THEN
      #bRunTrajectoryBtnEnbl := TRUE;
    ELSE
      #bRunTrajectoryBtnEnbl := FALSE;
    END_IF;

  ELSE
    #bAutoButtonsEnbl := FALSE;
    #bRunTrajectoryBtnEnbl := FALSE;
    #bShowSetpointAndError := FALSE;
    #bServiceBtnEnbl := TRUE;
    #bBtnService := FALSE;
  END_IF;

2: // MANUAL
  // Position buttons/sliders logic
  IF #bHeaveCompActive THEN
    #bShowSetpointAndError := TRUE;
    #bManualAndHeaveEnbl := TRUE;
  ELSE
    #bShowSetpointAndError := FALSE;
    #bManualAndHeaveEnbl := FALSE;
  END_IF;

  IF #bHeaveCompActive AND (#bSliderEnblSwitch OR #bExtEnblSwitch) THEN
    #bManualPosRef := TRUE;
    IF #bSliderEnblSwitch THEN
      #bExtEnblSwitch := FALSE;
      "fbStateMachine".rPayloadYRef := INT_TO_REAL(#nPayloadYRef);
    ELSE
      #bSliderEnblSwitch := FALSE;
      //"fbSystemController".rPayloadYRef := "rPotLeft":P;
      #rTestVariable := INT_TO_REAL("nPotLeft":P);
    END_IF;
  ELSE
    #bManualPosRef := FALSE;
    #bExtEnblSwitch := FALSE;
    #bSliderEnblSwitch := FALSE;
  END_IF;

  // Set jog dir
  "fbStateMachine".nJogDir := #nManualJogDir;

```

```
  3: // AUTOMATIC
    #bHeaveBtnEnbl := FALSE;
    #bRunTrajectoryBtnEnbl := FALSE;
    #bShowSetpointAndError := TRUE;
    #bManualBtnEnbl := FALSE;

    IF #bTrajectoryCompleted OR #bTrajectoryCanceled THEN
      #bTrajectoryGenerated := FALSE;
    END_IF;

  4: // SERVICE
    #bShowSetpointAndError := TRUE;
    #bHeaveBtnEnbl := FALSE;
    #bManualBtnEnbl := FALSE;

    IF #bServiceSetValues THEN
      "Parameters".rPosGainP := #rPositionGainP;
      "Parameters".tPosGainTi := DINT_TO_TIME(REAL_TO_DINT(#rPositionGainT *
1000.0));
      "Parameters".rVelGainP := #rVelocityGainP;
      "Parameters".tVelGainTi := DINT_TO_TIME(REAL_TO_DINT(#rVelocityGainT *
1000.0));
      #bServiceSetValues := FALSE;
    ELSIF #bServiceSetDefault THEN
      #rPositionGainP := "Parameters".rPosGainPDefault;
      #rPositionGainT := DINT_TO_REAL((TIME_TO_DINT("Parameters".tPosGainTiDe-
fault))) / 1000.0;
      #rVelocityGainP := "Parameters".rVelGainPDefault;
      #rVelocityGainT := DINT_TO_REAL((TIME_TO_DINT("Parameters".tVelGainTiDe-
fault))) / 1000.0;
      #bServiceSetDefault := FALSE;
    END_IF;
END_CASE;

```

Symbol	Address	Type	Comment
"DataInput".rFwDrum	%DB301.DBD16	Real	
"DataInput".rMotorTheta	%DB301.DBD8	Real	
"DataInput".rMotorThetad	%DB301.DBD12	Real	
"DataInput".rPayloadY	%DB301.DBD0	Real	
"DataOutput".rTorqueRef	%DB302.DBD0	Real	Torque reference for the motor.
"fbStateMachine".bHeaveCompActive	%DB2.DBX10.7	Bool	True if heave compensation is activated.
"fbStateMachine".bManualPosRef	%DB2.DBX11.1	Bool	True if position reference in manual mode.
"fbStateMachine".bSystemAutomatic	%DB2.DBX10.5	Bool	System automatic flag.
"fbStateMachine".bSystemEmg	%DB2.DBX10.1	Bool	Emergency flag.
"fbStateMachine".bSystemIdle	%DB2.DBX10.3	Bool	System idle flag
"fbStateMachine".bSystemManual	%DB2.DBX10.4	Bool	System manual flag.
"fbStateMachine".bSystemOn	%DB2.DBX10.2	Bool	System on flag.
"fbStateMachine".bSystemService	%DB2.DBX10.6	Bool	System service flag.
"fbStateMachine".bTrajectoryCanceled	%DB2.DBX32.1	Bool	True if trajectory is canceled.

Symbol	Address	Type	Comment
"fbStateMachine".bTrajectoryCompleted	%DB2.DBX32.0	Bool	True when trajectory is completed.
"fbStateMachine".bTrajectoryLandBtn	%DB2.DBX32.2	Bool	
"fbStateMachine".nCurrentState	%DB2.DBW12	Int	State machine current state.
"fbStateMachine".nJogDir	%DB2.DBW44	Int	Jog direction, 1, 0 or -1.
"fbStateMachine".nRequestState	%DB2.DBW14	Int	State machine requested state.
"fbStateMachine".rPayloadYdRef	%DB2.DBD40	Real	Payload velocity reference.
"fbStateMachine".rPayloadYRef	%DB2.DBD36	Real	Payload position reference.
"fbStateMachine".rTrajectoryTime	%DB2.DBD28	Real	Trajectory current time.
"fbStateMachine".rTrajectoryTp	%DB2.DBD24	Real	Trajectory time to complete.
"fbStateMachine".rTrajectoryYEnd	%DB2.DBD20	Real	Trajectory end position.
"fbStateMachine".sInfoMsg	P#DB2.DBX70.0	String	Info msg for hmi.
"GlobalVariables".bSystemTimeout	%DB16.DBX0.4	Bool	
"GlobalVariables".rThetaRef	%DB16.DBD18	Real	
"nPotLeft":P	%IW352:P	Int	
"Parameters".rDrumRadius	%DB8.DBD8	Real	Drum radius.
"Parameters".rGearRatio	%DB8.DBD0	Real	System gear ratio.
"Parameters".rPayloadMass	%DB8.DBD12	Real	Payload mass.
"Parameters".rPi	%DB8.DBD68	Real	Constant pi.
"Parameters".rPosGain	%DB8.DBD20	Real	Position controller gain.
"Parameters".rPosGainDefault	%DB8.DBD24	Real	Default position controller gain.
"Parameters".rSheaveRatio	%DB8.DBD4	Real	System sheave ratio.
"Parameters".rTravelingBlockMass	%DB8.DBD16	Real	Traveling block mass.
"Parameters".rVelGain	%DB8.DBD44	Real	Velocity controller gain.
"Parameters".rVelGainDefault	%DB8.DBD48	Real	Default velocity controller gain.
"Parameters".tPosGainTi	%DB8.DBD28	Time	Positioncontroller time.
"Parameters".tPosGainTiDefault	%DB8.DBD32	Time	Default position controller time.
"Parameters".tVelGainTi	%DB8.DBD52	Time	Velocity controller integration time.
"Parameters".tVelGainTiDefault	%DB8.DBD56	Time	Default velocity controller integration time.
#bAutoButtonsEnbl		Bool	Auto buttons are enabled if true.
#bBtnAutomatic		Bool	Button automatic.
#bBtnEmg		Bool	Button emergency.
#bBtnGenerateTrajectory		Bool	Button to generate trajectory.
#bBtnManual		Bool	Button manual.
#bBtnOn		Bool	Button on.
#bBtnOnEnbl		Bool	
#bBtnService		Bool	Button service.
#bExtEnblSwitch		Bool	Switch that enables or disables external position ref.

Symbol	Address	Type	Comment
#bFirstRunDone		Bool	Initializing flag.
#bHeaveBtnEnbl		Bool	True if heave comp button is active.
#bHeaveCompActive		Bool	Heave compensation active flag.
#bIdle		Bool	Idle state.
#bManualAndHeaveEnbl		Bool	True if in manual mode and ahc is enabled.
#bManualBtnEnbl		Bool	Manual button enabled if true.
#bManualPosRef		Bool	Manual position state in manual state (confusing? u stupid).
#bRunTrajectoryBtnEnbl		Bool	Run trajectory button is enabled if true.
#bServiceBtnEnbl		Bool	
#bServiceSetDefault		Bool	
#bServiceSetValues		Bool	
#bShowSetpointAndError		Bool	True if setpoint and error is visible.
#bSliderEnblSwitch		Bool	Switch that enables or disables position slider ref.
#bTimerDone		Bool	
#bTimerStart		Bool	
#bTrajectory5Btn		Bool	Predefined trajectory params to 5 button.
#bTrajectory75Btn		Bool	Predefined trajectory params to 7.5 button.
#bTrajectoryCanceled		Bool	
#bTrajectoryCompleted		Bool	
#bTrajectoryGenerated		Bool	True if trajectory is generated.
#bTrajectoryLandBtn		Bool	Predefined landing sequence button.
#bTrajectoryVisFalling		Bool	True if trajectory is negative (endpoint lower than current).
#nAnimPos		Int	Payload animation position as integer [0-100].
#nCurrentState		Int	System current state, from fbSystemController.
#nLastState		Int	System state last run.
#nManualJogDir		Int	1 if up is pressed, -1 if down is pressed.
#nPayloadYRef		Int	Manual mode payload position reference.
#nRequestedState		Int	System requested state.
#nTrendManualSelection		Int	
#rDrumRPM		Real	Drum speed [rpm].
#rHookLoad		Real	Hook load [kg].
#rK		Real	Conversion factor global to motor.
#rMotorPowerOut		Real	Motor power output.
#rMotorRPM		Real	Motor speed [rpm].
#rPayloadY		Real	Payload position in global frame.
#rPayloadYd		Real	Payload velocity in global frame.
#rPayloadYdFiltered		Real	
#rPayloadYdRef		Real	Manual mode payload velocity reference.
#rPayloadYError		Real	Payload position error.
#rPayloadYLast		Real	
#rPayloadYRef		Real	Payload position reference in global frame.
#rPlatformY		Real	Platform position in global frame.
#rPlatformY_moh		Real	Platform position in reference to sea level.
#rPlatformYd		Real	Platform velocity.
#rPositionGainP		Real	
#rPositionGainT		Real	
#rSeabedForce		Real	
#rTestVariable		Real	
#rTheta		Real	
#rThetaError		Real	
#rThetaRef		Real	
#rTorqueRef		Real	
#rTrajectoryAccVizLower		Real	

Symbol	Address	Type	Comment
#rTrajectoryAccVizUpper		Real	
#rTrajectoryTime		Real	
#rTrajectoryTimeLower-Limit		Real	
#rTrajectoryTp		Real	Time to complete trajectory.
#rTrajectoryVelViz		Real	
#rTrajectoryYEnd		Real	Trajectory endpoint.
#rTrendManualData		Real	
#rVelocityGainP		Real	
#rVelocityGainT		Real	
#rWireForce		Real	Wire force at drum [kN].
#sInfoMsg		String	Info string, shows up in top right corner of hmi.
#tTimerET		Time	

FB_StateMachine [FB2]

FB_StateMachine Properties					
General					
Name	FB_StateMachine	Number	2	Type	FB
Language	SCL	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input									
nLastCycleTime	Int	0.0	0	True	True	True	False		Last cycle time in milliseconds.
▼ Output									
rThetaRefOut	Real	2.0	0.0	True	True	True	False		
rThetadFFOut	Real	6.0	0.0	True	True	True	False		
InOut									
▼ Static									
bFirstRunDone	Bool	10.0	FALSE	True	True	True	False		
bSystemEmg	Bool	10.1	FALSE	True	True	True	False		Emergency flag.
bSystemOn	Bool	10.2	FALSE	True	True	True	False		System on flag.
bSystemIdle	Bool	10.3	FALSE	True	True	True	False		System idle flag
bSystemManual	Bool	10.4	FALSE	True	True	True	False		System manual flag.
bSystemAutomatic	Bool	10.5	FALSE	True	True	True	False		System automatic flag.
bSystemService	Bool	10.6	FALSE	True	True	True	False		System service flag.
bHeaveCompActive	Bool	10.7	FALSE	True	True	True	False		True if heave compensation is activated.
bHeaveCompMem	Bool	11.0	FALSE	True	True	True	False		Heave comp memory.
bManualPosRef	Bool	11.1	FALSE	True	True	True	False		True if position reference in manual mode.
nCurrentState	Int	12.0	#OFF	True	False	False	False		State machine current state.
nRequestState	Int	14.0	#OFF	True	False	False	False		State machine requested state.
rLastCycleTime	Real	16.0	0.0	True	True	True	False		Last cycle time in seconds.

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
rTrajectoryYEnd	Real	20.0	0.0	True	True	True	False		Trajectory end position.
rTrajectoryTp	Real	24.0	0.0	True	True	True	False		Trajectory time to complete.
rTrajectoryTime	Real	28.0	0.0	True	False	False	False		Trajectory current time.
bTrajectoryCompleted	Bool	32.0	FALSE	True	True	True	False		True when trajectory is completed.
bTrajectoryCanceled	Bool	32.1	FALSE	True	True	True	False		True if trajectory is canceled.
bTrajectoryLandBtn	Bool	32.2	FALSE	True	True	True	False		
nLandingSequence	Int	34.0	0	True	True	True	False		Initiate landing sequence.
rPayloadYRef	Real	36.0	7.5	True	True	True	False		Payload position reference.
rPayloadYdRef	Real	40.0	0.0	True	True	True	False		Payload velocity reference.
nJogDir	Int	44.0	0	True	True	True	False		Jog direction, 1, 0 or -1.
rThetadPayloadFF	Real	46.0	0.0	True	True	True	False		Motor angular velocity ff from payload
rThetadPlatformFF	Real	50.0	0.0	True	True	True	False		Motor angular velocity ff from platform
rThetaddPlatformFF	Real	54.0	0.0	True	True	True	False		Motor angular acceleration ff from platform.
rThetaRef	Real	58.0	-45175.0	True	True	True	False		Motor angle reference.
rThetadFF	Real	62.0	0.0	True	True	True	False		Motor angular velocity feedforward.
rThetaddFF	Real	66.0	0.0	True	True	True	False		Motor angular acceleration feedforward.
sInfoMsg	String	70.0	''	True	True	True	False		Info msg for hmi.
rTensionRef	Real	326.0	25500.0	True	True	True	False		
▼ Temp									
rPayloadYTrajRef	Real	0.0							
rPayloadYdTrajRef	Real	4.0							
rPayloadYdLocal	Real	8.0							Current payload velocity local frame (platform/motor frame).
▼ Constant									
OFF	Int		0						
IDLE	Int		1						
MANUAL	Int		2						
AUTOMATIC	Int		3						
SERVICE	Int		4						

```
// Initialize
IF NOT #bFirstRunDone THEN
  #bFirstRunDone := TRUE;
END_IF;

// Convert last cycle time to seconds
#rLastCycleTime := INT_TO_REAL(#nLastCycleTime)*(10**(-3));

// Check variables and request state change
IF #bSystemEmg OR "GlobalVariables".bSystemTimeout THEN
  #nRequestState := #OFF;
ELSIF #nCurrentState = #OFF AND #bSystemOn THEN
  #nRequestState := #IDLE;
ELSIF #nCurrentState = #IDLE AND NOT #bSystemOn THEN
  #nRequestState := #OFF;
ELSIF #nCurrentState <> #IDLE AND #bSystemIdle THEN
  #nRequestState := #IDLE;
ELSIF #nCurrentState = #IDLE AND #bSystemService THEN
  #nRequestState := #SERVICE;
ELSIF #nCurrentState = #SERVICE AND NOT #bSystemService THEN
  #nRequestState := #IDLE;
ELSIF #nCurrentState = #IDLE AND #bSystemManual THEN
  #nRequestState := #MANUAL;
ELSIF #nCurrentState = #IDLE AND #bSystemAutomatic THEN
  #nRequestState := #AUTOMATIC;
ELSIF #nCurrentState = #MANUAL AND NOT #bSystemManual THEN
  #nRequestState := #IDLE;
ELSIF #nCurrentState = #AUTOMATIC AND NOT #bSystemAutomatic THEN
  #nRequestState := #IDLE;
END_IF;

// Check if requested state change is allowed
IF #nRequestState <> #nCurrentState THEN
  CASE #nCurrentState OF
    #OFF: // 0
      IF #nRequestState = #IDLE THEN
        "F_SetState"(#IDLE);
      END_IF;

    #IDLE: // 1
      IF #nRequestState = #OFF THEN
        "F_SetState"(#OFF);
      ELSIF #nRequestState = #MANUAL THEN
        "F_SetState"(#MANUAL);
      ELSIF #nRequestState = #AUTOMATIC THEN
        "F_SetState"(#AUTOMATIC);
      ELSIF #nRequestState = #SERVICE THEN
        "F_SetState"(#SERVICE);
      END_IF;

    #MANUAL: // 2
      IF #nRequestState = #OFF THEN
        "F_SetState"(#OFF);
      ELSIF #nRequestState = #IDLE THEN
        "F_SetState"(#IDLE);
      ELSIF #nRequestState = #AUTOMATIC THEN
        "F_SetState"(#AUTOMATIC);
      END_IF;
```

```
    #AUTOMATIC: // 3
      IF #nRequestState = #OFF THEN
        "F_SetState"(#OFF);
      ELSIF #nRequestState = #IDLE THEN
        "F_SetState"(#IDLE);
      END_IF;

    #SERVICE: // 4
      IF #nRequestState = #OFF THEN
        "F_SetState"(#OFF);
      ELSIF #nRequestState = #IDLE THEN
        "F_SetState"(#IDLE);
      END_IF;

  END_CASE;
END_IF;


// Check if state did not change
IF #nCurrentState <> #nRequestState THEN
  #nRequestState := #nCurrentState;
END_IF;

// Check if AHC has been changed
IF #bHeaveCompActive <> #bHeaveCompMem THEN
  IF #bHeaveCompActive THEN
    #sInfoMsg := 'Heave comp activated';
    #rPayloadYRef := "DataInput".rPayloadY;
    "DataOutput".rBrakeDisabled := 1.0;
  ELSE
    #sInfoMsg := 'Heave comp deactivated';
    #rThetaRef := "DataInput".rMotorTheta;
    "DataOutput".rBrakeDisabled := 0.0;
  END_IF;
  #bHeaveCompMem := #bHeaveCompActive;
END_IF;

// Run state machine
CASE #nCurrentState OF
  #OFF: // 0
    #rThetadPayloadFF := 0.0;
    #rThetadPlatformFF := 0.0;
    "GlobalVariables".bMainCtrlSystemEnbl := FALSE;
    "GlobalVariables".bTensionCtrlSystemEnbl := FALSE;

  #IDLE: // 1
    "GlobalVariables".bMainCtrlSystemEnbl := TRUE;

    IF #bHeaveCompActive THEN
      #rThetaRef := (#rPayloadYRef - "DataInput".rPayloadY) * "fbDataProcessing".rK + "DataInput".rMotorTheta;
      #rThetadPayloadFF := 0;
      #rThetadPlatformFF := "fbDataProcessing".rPlatformYdFiltered * "fbDataProcessing".rK;
    ELSE
      #rThetadPayloadFF := 0.0;
      #rThetadPlatformFF := 0.0;
    END_IF;

```

```
 #MANUAL: // 2
   IF #bManualPosRef THEN // Ahc must be active, payload position reference
     #rThetaRef := (#rPayloadYRef - "DataInput".rPayloadY) * "fbDataProcessing".rK + "DataInput".rMotorTheta;
     #rThetadPayloadFF := 0.0;
     #rThetadPlatformFF := "fbDataProcessing".rPlatformYdFiltered * "fbDataProcessing".rK;

   ELSIF #bHeaveCompActive THEN // If ahc is active, payload velocity reference in global frame
     // Ramp velocity reference
     "fbRamp"(rEndValue := "Parameters".rJogVelocity * #nJogDir,
         rRampTime := "Parameters".rJogRampTime,
         rCurrentValue := #rPayloadYdRef,
         rLastCycleTime := #rLastCycleTime,
         rReference := #rPayloadYdRef);

     // Calculate motor references and feedforwards
     #rPayloadYRef := #rPayloadYRef + #rPayloadYdRef* #rLastCycleTime;

     // Saturate payload reference at endpoints
     IF #rPayloadYRef > "Parameters".rPayloadYMax THEN
       #rPayloadYRef := "Parameters".rPayloadYMax;
       #rPayloadYdRef := 0.0;
     ELSIF #rPayloadYRef < "Parameters".rPayloadYMin THEN
       #rPayloadYRef := "Parameters".rPayloadYMin;
       #rPayloadYdRef := 0.0;
     END_IF;

     // Calculate motor references and feedforwards
     #rThetaRef := (#rPayloadYRef - "DataInput".rPayloadY) * "fbDataProcessing".rK + "DataInput".rMotorTheta;
     #rThetadPayloadFF := #rPayloadYdRef * "fbDataProcessing".rK;
     #rThetadPlatformFF := "fbDataProcessing".rPlatformYdFiltered * "fbDataProcessing".rK;

   ELSE
     // Ramp velocity reference
     "fbRamp"(rEndValue := "Parameters".rJogVelocity * #nJogDir,
         rRampTime := "Parameters".rJogRampTime,
         rCurrentValue := #rPayloadYdRef,
         rLastCycleTime := #rLastCycleTime,
         rReference := #rPayloadYdRef);

     // Calculate motor references and feedforwards
     #rThetaRef := #rThetaRef + #rThetadFF * #rLastCycleTime;
     #rThetadPayloadFF := #rPayloadYdRef * "fbDataProcessing".rK;
     #rThetadPlatformFF := 0.0;
   END_IF;

 #AUTOMATIC: // 3
   // Increment trajectory time
   #rTrajectoryTime := #rTrajectoryTime + #rLastCycleTime;

   IF #bTrajectoryLandBtn THEN
     #nLandingSequence := 1;
     #bTrajectoryLandBtn := FALSE;
   END_IF;

```

```
IF #nLandingSequence > 0 THEN
  // Run landing sequence
  CASE #nLandingSequence OF
    1: // Interpolate down to 0.0m
      "fbQuintic"(rLastCycleTime := #rLastCycleTime,
            rYRef => #rPayloadYRef,
            rYdRef => #rPayloadYdRef,
            bTrajectoryCompleted => #bTrajectoryCompleted);

      IF #bTrajectoryCompleted THEN
        #nLandingSequence := 2;
        #bTrajectoryCompleted := FALSE;
      END_IF;

    2: // Initiate constant tension controller
      "GlobalVariables".bMainCtrlSystemEnbl := FALSE;
      "GlobalVariables".bTensionCtrlSystemEnbl := TRUE;

      "fbRamp"(rEndValue := 1471.5,
           rRampTime := 9.5,
           rCurrentValue := #rTensionRef,
           rLastCycleTime := #rLastCycleTime,
           rReference := #rTensionRef);

      "GlobalVariables".rTensionRef := #rTensionRef;

      IF #rTensionRef <= 1471.5 THEN
        "GlobalVariables".rTensionRef := 1471.5;
      ELSE
        "GlobalVariables".rTensionRef := #rTensionRef;
      END_IF;

    3:
      "GlobalVariables".bMainCtrlSystemEnbl := TRUE;
      "GlobalVariables".bTensionCtrlSystemEnbl := FALSE;
      #rPayloadYRef := 1.0;
      #bTrajectoryCompleted := TRUE;
  END_CASE;

ELSE
  "fbQuintic"(rLastCycleTime := #rLastCycleTime,
        rYRef => #rPayloadYRef,
        rYdRef => #rPayloadYdRef,
        bTrajectoryCompleted => #bTrajectoryCompleted);
END_IF;

// Reset to IDLE if trajectory is completed or canceled
IF #bTrajectoryCompleted OR #bTrajectoryCanceled THEN
  #nRequestState := #IDLE;
  #nLandingSequence := 0;
  #bTrajectoryCanceled := FALSE;
  #bTrajectoryCompleted := FALSE;
  #bSystemAutomatic := FALSE;
  #rTrajectoryTime := 0.0;
  "fbQuintic".bReset := TRUE;
END_IF;

// Calculate motor references and feedforwards
```

```
    #rThetaRef := (#rPayloadYRef - "DataInput".rPayloadY) * "fbDataProcessing".rK + "DataInput".rMotorTheta;
    #rThetadPayloadFF := #rPayloadYdRef * "fbDataProcessing".rK;
    #rThetadPlatformFF := "fbDataProcessing".rPlatformYdFiltered * "fbDataProcessing".rK;

  #SERVICE: // 4
    #rThetaRef := (#rPayloadYRef - "DataInput".rPayloadY) * "fbDataProcessing".rK + "DataInput".rMotorTheta;
    #rThetadPayloadFF := 0;
    #rThetadPlatformFF := "fbDataProcessing".rPlatformYdFiltered * "fbDataProcessing".rK;
END_CASE;

// Calculate feedforwards
#rThetadFF := #rThetadPayloadFF - #rThetadPlatformFF;

// Assign outputs
IF #bHeaveCompActive THEN
  #rThetaRefOut := #rThetaRef;
  #rThetadFFOut := #rThetadFF;
ELSE
  #rThetaRefOut := "DataInput".rMotorTheta;
  #rThetadFFOut := 0.0;
END_IF;

```

Symbol	Address	Type	Comment
"DataInput".rMotorTheta	%DB301.DBD8	Real	
"DataInput".rPayloadY	%DB301.DBD0	Real	
"DataOutput".rBrakeDisabled	%DB302.DBD4	Real	If true, the brake is disabled.
"fbDataProcessing".rK	%DB7.DBD8	Real	
"fbDataProcessing".rPlatformYdFiltered	%DB7.DBD24	Real	
"fbQuintic".bReset	%DB10.DBX30.0	Bool	
"GlobalVariables".bMainCtrlSystemEnbl	%DB16.DBX26.0	Bool	
"GlobalVariables".bSystemTimeout	%DB16.DBX0.4	Bool	
"GlobalVariables".bTensionCtrlSystemEnbl	%DB16.DBX26.1	Bool	
"GlobalVariables".rTensionRef	%DB16.DBD28	Real	
"Parameters".rJogRampTime	%DB8.DBD72	Real	Ramp time to rJogVelocity.
"Parameters".rJogVelocity	%DB8.DBD76	Real	Max jog velocity (abs value).
"Parameters".rPayloadMax	%DB8.DBD88	Real	Payload maximum position, soft limit.
"Parameters".rPayloadMin	%DB8.DBD84	Real	Payload minimum position, soft limit.
#AUTOMATIC	3	Int	
#bFirstRunDone		Bool	
#bHeaveCompActive		Bool	True if heave compensation is activated.
#bHeaveCompMem		Bool	Heave comp memory.
#bManualPosRef		Bool	True if position reference in manual mode.
#bSystemAutomatic		Bool	System automatic flag.
#bSystemEmg		Bool	Emergency flag.

Symbol	Address	Type	Comment
#bSystemIdle		Bool	System idle flag
#bSystemManual		Bool	System manual flag.
#bSystemOn		Bool	System on flag.
#bSystemService		Bool	System service flag.
#bTrajectoryCanceled		Bool	True if trajectory is canceled.
#bTrajectoryCompleted		Bool	True when trajectory is completed.
#bTrajectoryLandBtn		Bool	
#IDLE	1	Int	
#MANUAL	2	Int	
#nCurrentState		Int	State machine current state.
#nJogDir		Int	Jog direction, 1, 0 or -1.
#nLandingSequence		Int	Initiate landing sequence.
#nLastCycleTime		Int	Last cycle time in milliseconds.
#nRequestState		Int	State machine requested state.
#OFF	0	Int	
#rLastCycleTime		Real	Last cycle time in seconds.
#rPayloadYdRef		Real	Payload velocity reference.
#rPayloadYRef		Real	Payload position reference.
#rTensionRef		Real	
#rThetadFF		Real	Motor angular velocity feedforward.
#rThetadFFOut		Real	
#rThetadPayloadFF		Real	Motor angular velocity ff from payload
#rThetadPlatformFF		Real	Motor angular velocity ff from platform
#rThetaRef		Real	Motor angle reference.
#rThetaRefOut		Real	
#rTrajectoryTime		Real	Trajectory current time.
#SERVICE	4	Int	
#sInfoMsg		String	Info msg for hmi.

F.2 Data blocks

DataInput [DB301]

DataInput Properties					
General					
Name	DataInput	Number	301	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Static										
rPayloadY	Real	0.0	0.0	True	True	True	True	False		
rPlatformYdd	Real	4.0	0.0	True	True	True	True	False		
rMotorTheta	Real	8.0	0.0	True	True	True	True	False		
rMotorThetad	Real	12.0	0.0	True	True	True	True	False		
rFwDrum	Real	16.0	0.0	True	True	True	True	False		
rPower	Real	20.0	0.0	True	True	True	True	False		
rHeartbeat	Real	24.0	0.0	True	True	True	True	False		

DataOutput [DB302]

DataOutput Properties					
General					
Name	DataOutput	Number	302	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Static										
rTorqueRef	Real	0.0	0.0	True	True	True	True	False		Torque reference for the motor.
rBrakeDisabled	Real	4.0	0.0	True	True	True	True	False		If true, the brake is disabled.

fbDataProcessing [DB7]

fbDataProcessing Properties					
General					
Name	fbDataProcessing	Number	7	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input										
rPlatformYdd	Real	0.0	0.0	True	True	True	True	False		
nLastCycleTime	Int	4.0	0	True	True	True	True	False		
Output										
InOut										
▼ Static										
bFirstRunDone	Bool	6.0	false	True	True	True	True	False		
rK	Real	8.0	0.0	True	True	True	True	False		
rMotorThetad_p	Real	12.0	0.0	True	True	True	True	False		
rMotorThetad_p	Real	16.0	0.0	True	True	True	True	False		
rPlatformYd	Real	20.0	0.0	True	True	True	True	False		
rPlatformYdFiltered	Real	24.0	0.0	True	True	True	True	False		
rLastycleTime	Real	28.0	0.0	True	True	True	True	False		

fbLowPassFilterPayloadYd [DB17]

fbLowPassFilterPayloadYd Properties					
General					
Name	fbLowPassFilterPayloadYd	Number	17	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input										
rCutoffFrequency	Real	0.0	100.0	True	True	True	True	False		Break frequency.
rDt	Real	4.0	1.0	True	True	True	True	False		Cycle time in seconds.
rX	Real	8.0	0.0	True	True	True	True	False		Input value.
▼ Output										
rY	Real	12.0	0.0	True	True	True	True	False		Output value.
InOut										
▼ Static										
rYLast	Real	16.0	0.0	True	True	True	True	False		Last output value.
rOmega	Real	20.0	0.0	True	True	True	True	False		Angular frequency.

fbLowPassFilterPlatformYd [DB11]

fbLowPassFilterPlatformYd Properties					
General					
Name	fbLowPassFilterPlatformYd	Number	11	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input										
rCutoffFrequency	Real	0.0	100.0	True	True	True	True	False		Break frequency.
rDt	Real	4.0	1.0	True	True	True	True	False		Cycle time in seconds.
rX	Real	8.0	0.0	True	True	True	True	False		Input value.
▼ Output										
rY	Real	12.0	0.0	True	True	True	True	False		Output value.
InOut										
▼ Static										
rYLast	Real	16.0	0.0	True	True	True	True	False		Last output value.
rOmega	Real	20.0	0.0	True	True	True	True	False		Angular frequency.

fbManagementHMI [DB18]

fbManagementHMI Properties					
General					
Name	fbManagementHMI	Number	18	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
Input										
Output										
InOut										
▼ Static										
rPowerUsage	Real	0.0	0.0	True	True	True	True	False		
rEnergyUsage	Real	4.0	0.0	True	True	True	True	False		
rDowntime	Real	8.0	0.0	True	True	True	True	False		
rUptime	Real	12.0	0.0	True	True	True	True	False		
rPayloadY	Real	16.0	7.5	True	True	True	True	False		
rPlatformY	Real	20.0	350.0	True	True	True	True	False		

fbOperatorHMI [DB12]

fbOperatorHMI Properties					
General					
Name	fbOperatorHMI	Number	12	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
Input										
Output										
InOut										
▼ Static										
bFirstRunDone	Bool	0.0	FALSE	True	True	True	True	False		Initializing flag.
bBtnOn	Bool	0.1	FALSE	True	True	True	True	False		Button on.
bBtnOnEnbl	Bool	0.2	TRUE	True	True	True	True	False		
bBtnManual	Bool	0.3	FALSE	True	True	True	True	False		Button manual.
bManualBtnEnbl	Bool	0.4	FALSE	True	True	True	True	False		Manual button enabled if true.
bBtnAutomatic	Bool	0.5	FALSE	True	True	True	True	False		Button automatic.
bAutoButtonsEnbl	Bool	0.6	FALSE	True	True	True	True	False		Auto buttons are enabled if true.
bBtnEmg	Bool	0.7	FALSE	True	True	True	True	False		Button emergency.
bIdle	Bool	1.0	FALSE	True	True	True	True	False		Idle state.
bShowSetpointAndError	Bool	1.1	FALSE	True	True	True	True	False		True if setpoint and error is visible.
bBtnService	Bool	1.2	FALSE	True	True	True	True	False		Button service.
bServiceBtnEnbl	Bool	1.3	FALSE	True	True	True	True	False		
rPositionGainP	Real	2.0	1.0	True	True	True	True	False		
rPositionGainT	Real	6.0	0.0	True	True	True	True	False		
rVelocityGainP	Real	10.0	1.0	True	True	True	True	False		
rVelocityGainT	Real	14.0	1.0	True	True	True	True	False		
bServiceSetValues	Bool	18.0	FALSE	True	True	True	True	False		

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
bServiceSetDefault	Bool	18.1	TRUE	True	True	True	True	False		
bSliderEnblSwitch	Bool	18.2	FALSE	True	True	True	True	False		Switch that enables or disables position slider ref.
bManualAndHeaveEnbl	Bool	18.3	FALSE	True	True	True	True	False		True if in manual mode and ahc is enabled.
bExtEnblSwitch	Bool	18.4	FALSE	True	True	True	True	False		Switch that enables or disables external position ref.
bManualPosRef	Bool	18.5	FALSE	True	True	True	True	False		Manual position state in manual state (confusing? u stupid).
bHeaveCompActive	Bool	18.6	FALSE	True	True	True	True	False		Heave compensation active flag.
bHeaveBtnEnbl	Bool	18.7	FALSE	True	True	True	True	False		True if heave comp button is active.
nAnimPos	Int	20.0	0	True	True	True	True	False		Payload animation position as integer [0-100].
rPlatformY	Real	22.0	350.0	True	True	True	True	False		Platform position in global frame.
rPlatformY_moh	Real	26.0	0.0	True	True	True	True	False		Platform position in reference to sea level.
rPlatformYd	Real	30.0	0.0	True	True	True	True	False		Platform velocity.
rPayloadY	Real	34.0	7.5	True	True	True	True	False		Payload position in global frame.
rPayloadYLast	Real	38.0	7.5	True	True	True	True	False		
rPayloadYRef	Real	42.0	7.5	True	True	True	True	False		Payload position reference in global frame.
rPayloadYError	Real	46.0	0.0	True	True	True	True	False		Payload position error.
rPayloadYd	Real	50.0	0.0	True	True	True	True	False		Payload velocity in global frame.
rPayloadYdFiltered	Real	54.0	0.0	True	True	True	True	False		
rMotorPowerOut	Real	58.0	0.0	True	True	True	True	False		Motor power output.
nPayloadYRef	Int	62.0	7	True	True	True	True	False		Manual mode payload position reference.
rPayloadYdRef	Real	64.0	7.5	True	True	True	True	False		Manual mode payload velocity reference.
nManualJogDir	Int	68.0	0	True	True	True	True	False		1 if up is pressed, -1 if down is pressed.
rMotorRPM	Real	70.0	0.0	True	True	True	True	False		Motor speed [rpm].
rDrumRPM	Real	74.0	0.0	True	True	True	True	False		Drum speed [rpm].
rTheta	Real	78.0	0.0	True	True	True	True	False		

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
rThetaRef	Real	82.0	0.0	True	True	True	True	False		
rThetaError	Real	86.0	0.0	True	True	True	True	False		
rWireForce	Real	90.0	0.0	True	True	True	True	False		Wire force at drum [kN].
rSeabedForce	Real	94.0	0.0	True	True	True	True	False		
rHookLoad	Real	98.0	0.0	True	True	True	True	False		Hook load [kg].
bBtnGenerateTrajectory	Bool	102.0	FALSE	True	True	True	True	False		Button to generate trajectory.
bTrajectoryGenerated	Bool	102.1	TRUE	True	True	True	True	False		True if trajectory is generated.
bRunTrajectoryBtnEnbl	Bool	102.2	FALSE	True	True	True	True	False		Run trajectory button is enabled if true.
bTrajectory75Btn	Bool	102.3	FALSE	True	True	True	True	False		Predefined trajectory params to 7.5 button.
bTrajectory5Btn	Bool	102.4	FALSE	True	True	True	True	False		Predefined trajectory params to 5 button.
bTrajectoryLandBtn	Bool	102.5	FALSE	True	True	True	True	False		Predefined landing sequence button.
nLandingSequence	Int	104.0	0	True	True	True	True	False		Initiate landing sequence.
rTrajectoryVelVz	Real	106.0	0.0	True	True	True	True	False		
rTrajectoryAccVrzUpper	Real	110.0	0.0	True	True	True	True	False		
rTrajectoryAccVrzLower	Real	114.0	0.0	True	True	True	True	False		
bTrajectoryCompleted	Bool	118.0	FALSE	True	True	True	True	False		
bTrajectoryCanceled	Bool	118.1	FALSE	True	True	True	True	False		
bTrajectoryVisFalling	Bool	118.2	TRUE	True	True	True	True	False		True if trajectory is negative (endpoint lower than current).
rTrajectoryYEnd	Real	120.0	1.0	True	True	True	True	False		Trajectory endpoint.
rTrajectoryTp	Real	124.0	5.0	True	True	True	True	False		Time to complete trajectory.
rTrajectoryTime	Real	128.0	10.0	True	True	True	True	False		
rK	Real	132.0	1.0	True	True	False	False	False		Conversion factor global to motor.
nCurrentState	Int	136.0	0	True	True	True	True	False		System current state, from fbSystemController.
nRequestedState	Int	138.0	0	True	True	True	True	False		System requested state.
nLastState	Int	140.0	0	True	True	True	True	False		System state last run.

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
sInfoMsg	String	142.0	''	True	True	True	True	False		Info string, shows up in top right corner of hmi.
bTimerStart	Bool	398.0	TRUE	True	True	True	True	False		
bTimerDone	Bool	398.1	false	True	True	True	True	False		
tTimerET	Time	400.0	T#0ms	True	True	True	True	False		
rTestVariable	Real	404.0	0.0	True	True	True	True	False		
nTrendManualSelection	Int	408.0	0	True	True	True	True	False		
rTrendManualData	Real	410.0	0.0	True	True	True	True	False		
rTorqueRef	Real	414.0	0.0	True	True	True	True	False		

fbQuintic [DB10]

fbQuintic Properties					
General					
Name	fbQuintic	Number	10	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input										
rLastCycleTime	Real	0.0	0.0	True	True	True	True	False		Last cycle time.
▼ Output										
rYRef	Real	4.0	0.0	True	True	True	True	False		
rYdRef	Real	8.0	0.0	True	True	True	True	False		
bTrajectoryCompleted	Bool	12.0	false	True	True	True	True	False		
InOut										
▼ Static										
rT	Real	14.0	0.0	True	True	True	True	False		
rTp	Real	18.0	0.0	True	True	False	False	False		
rY0	Real	22.0	0.0	True	True	False	False	False		
rY1	Real	26.0	0.0	True	True	False	False	False		
bReset	Bool	30.0	false	True	True	True	True	False		

fbRamp [DB13]

fbRamp Properties					
General					
Name	fbRamp	Number	13	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input										
rEndValue	Real	0.0	0.0	True	True	True	True	False		Desired end value.
rRampTime	Real	4.0	0.0	True	True	True	True	False		Desired ramp time.
rCurrentValue	Real	8.0	0.0	True	True	True	True	False		Current value.
rLastCycleTime	Real	12.0	0.0	True	True	True	True	False		Last cycle time (dt).
Output										
▼ InOut										
rReference	Real	16.0	0.0	True	True	True	True	False		Reference inout.
▼ Static										
rSlope	Real	20.0	0.0	True	True	True	True	False		Ramp (dy/dt).
rEndValueMem	Real	24.0	0.0	True	True	True	True	False		End value memory.
rTolerance	Real	28.0	0.1	True	True	True	True	False		Preliminary tolerance.
bTrajectoryCompleted	Bool	32.0	false	True	True	True	True	False		

fbStateMachine [DB2]

fbStateMachine Properties					
General					
Name	fbStateMachine	Number	2	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input										
nLastCycleTime	Int	0.0	0	True	True	True	True	False		Last cycle time in milliseconds.
▼ Output										
rThetaRefOut	Real	2.0	0.0	True	True	True	True	False		
rThetadFFOut	Real	6.0	0.0	True	True	True	True	False		
InOut										
▼ Static										
bFirstRunDone	Bool	10.0	FALSE	True	True	True	True	False		
bSystemEmg	Bool	10.1	FALSE	True	True	True	True	False		Emergency flag.
bSystemOn	Bool	10.2	FALSE	True	True	True	True	False		System on flag.
bSystemIdle	Bool	10.3	FALSE	True	True	True	True	False		System idle flag
bSystemManual	Bool	10.4	FALSE	True	True	True	True	False		System manual flag.
bSystemAutomatic	Bool	10.5	FALSE	True	True	True	True	False		System automatic flag.
bSystemService	Bool	10.6	FALSE	True	True	True	True	False		System service flag.
bHeaveCompActive	Bool	10.7	FALSE	True	True	True	True	False		True if heave compensation is activated.
bHeaveCompMem	Bool	11.0	FALSE	True	True	True	True	False		Heave comp memory.
bManualPosRef	Bool	11.1	FALSE	True	True	True	True	False		True if position reference in manual mode.
nCurrentState	Int	12.0	0	True	True	False	False	False		State machine current state.
nRequestState	Int	14.0	0	True	True	False	False	False		State machine requested state.
rLastCycleTime	Real	16.0	0.0	True	True	True	True	False		Last cycle time in seconds.
rTrajectoryYEnd	Real	20.0	0.0	True	True	True	True	False		Trajectory end position.

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
rTrajectoryTp	Real	24.0	0.0	True	True	True	True	False		Trajectory time to complete.
rTrajectoryTime	Real	28.0	0.0	True	True	False	False	False		Trajectory current time.
bTrajectoryCompleted	Bool	32.0	FALSE	True	True	True	True	False		True when trajectory is completed.
bTrajectoryCanceled	Bool	32.1	FALSE	True	True	True	True	False		True if trajectory is canceled.
bTrajectoryLandBtn	Bool	32.2	FALSE	True	True	True	True	False		
nLandingSequence	Int	34.0	0	True	True	True	True	False		Initiate landing sequence.
rPayloadYRef	Real	36.0	7.5	True	True	True	True	False		Payload position reference.
rPayloadYdRef	Real	40.0	0.0	True	True	True	True	False		Payload velocity reference.
nJogDir	Int	44.0	0	True	True	True	True	False		Jog direction, 1, 0 or -1.
rThetadPayloadFF	Real	46.0	0.0	True	True	True	True	False		Motor angular velocity ff from payload
rThetadPlatformFF	Real	50.0	0.0	True	True	True	True	False		Motor angular velocity ff from platform
rThetaddPlatformFF	Real	54.0	0.0	True	True	True	True	False		Motor angular acceleration ff from platform.
rThetaRef	Real	58.0	-45175.0	True	True	True	True	False		Motor angle reference.
rThetadFF	Real	62.0	0.0	True	True	True	True	False		Motor angular velocity feedforward.
rThetaddFF	Real	66.0	0.0	True	True	True	True	False		Motor angular acceleration feedforward.
sInfoMsg	String	70.0	''	True	True	True	True	False		Info msg for hmi.
rTensionRef	Real	326.0	25500.0	True	True	True	True	False		

GlobalVariables [DB16]

GlobalVariables Properties					
General					
Name	GlobalVariables	Number	16	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Static										
bSendData	Bool	0.0	FALSE	True	True	True	True	False		Sendt data flag. Is set high every 10ms, and low right after.
bSystemTimeoutPosTrig	Bool	0.1	false	True	True	True	True	False		
bSystemTimoutNegTrig	Bool	0.2	false	True	True	True	True	False		
bSystemTimoutSetReset	Bool	0.3	false	True	True	True	True	False		
bSystemTimeout	Bool	0.4	false	True	True	True	True	False		
tSystemUptime	Time	2.0	T#0ms	True	True	True	True	False		
tSystemDowntime	Time	6.0	T#0ms	True	True	True	True	False		
tSystemOnTime	Time	10.0	T#0ms	True	True	True	True	False		
rEnergyUsage	Real	14.0	0.0	True	True	True	True	False		
rThetaRef	Real	18.0	0.0	True	True	True	True	False		
rThetadFF	Real	22.0	0.0	True	True	True	True	False		
bMainCtrlSystemEnbl	Bool	26.0	FALSE	True	True	True	True	False		
bTensionCtrlSystemEnbl	Bool	26.1	TRUE	True	True	True	True	False		
rTensionRef	Real	28.0	0.0	True	True	True	True	False		

Parameters [DB8]

Parameters Properties					
General					
Name	Parameters	Number	8	Type	DB
Language	DB	Numbering	Automatic		
Information					
Title		Author		Comment	
Family		Version	0.1	User-defined ID	

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Static										
rGearRatio	Real	0.0	4.5	True	True	True	True	False		System gear ratio.
rSheaveRatio	Real	4.0	4.0	True	True	True	True	False		System sheave ratio.
rDrumRadius	Real	8.0	0.11615	True	True	True	True	False		Drum radius.
rPayloadMass	Real	12.0	12000.0	True	True	True	True	False		Payload mass.
rTravelingBlockMass	Real	16.0	600.0	True	True	True	True	False		Traveling block mass.
rPosGainP	Real	20.0	15.0	True	True	True	True	False		Position controller gain.
rPosGainPDefault	Real	24.0	15.0	True	True	True	True	False		Default position controller gain.
tPosGainTi	Time	28.0	T#3s	True	True	True	True	False		Positioncontroller time.
tPosGainTiDefault	Time	32.0	T#3s	True	True	True	True	False		Default position controller time.
rPosSatDefault	Real	36.0	157.0	True	True	True	True	False		Default position controller saturation.
rPosSatDefaultNeg	Real	40.0	-157.0	True	True	True	True	False		Default negative position controller saturation.
rVelGainP	Real	44.0	88.0	True	True	True	True	False		Velocity controller gain.
rVelGainPDefault	Real	48.0	88.0	True	True	True	True	False		Default velocity controller gain.
tVelGainTi	Time	52.0	T#12s	True	True	True	True	False		Velocity controller integration time.
tVelGainTiDefault	Time	56.0	T#12s	True	True	True	True	False		Default velocity controller integration time.
rVelSatDefault	Real	60.0	1600.0	True	True	True	True	False		Default velocity controller saturation.
rVelSatDefualtNeg	Real	64.0	-1600.0	True	True	True	True	False		Default negative velocity controller saturation.
rPi	Real	68.0	3.141593	True	True	False	True	False		Constant pi.

Name	Data type	Offset	Start value	Retain	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
rJogRampTime	Real	72.0	2.0	True	True	True	True	False		Ramp time to rJogVelocity.
rJogVelocity	Real	76.0	1.0	True	True	True	True	False		Max jog velocity (abs value).
tSystemTimout	Time	80.0	T#2s	True	True	True	True	False		Time from last message to system timeout.
rPayloadYMin	Real	84.0	1.0	True	True	True	True	False		Payload minimum position, soft limit.
rPayloadYMax	Real	88.0	14.0	True	True	True	True	False		Payload maximum position, soft limit.
rTensionGainP	Real	92.0	0.5	True	True	True	True	False		
tTensionGainTi	Time	96.0	T#1.5s	True	True	True	True	False		

F.3 Misc Functions & Function Blocks

F_QuinticMaxAcc [FC3]

F_QuinticMaxAcc Properties					
General					
Name	F_QuinticMaxAcc	**Number**	3	**Type**	FC
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Super-vision	Comment
▼ Input					
rY0	Real				Starting point.
rY1	Real				Ending point.
rTp	Real				Trajectory time.
Output					
InOut					
▼ Temp					
rTa	Real	0.0			
Constant					
▼ Return					
F_QuinticMaxAcc	Real				

```
#rTa := (#rTp * (SQRT(3) - 3) / (6));
#F_QuinticMaxAcc := (180 * #rTa ** 2 * (#rY0 - #rY1)) / #rTp ** 4 - (60 * #rTa
 * (#rY0 - #rY1)) / #rTp ** 3 - (120 * #rTa ** 3 * (#rY0 - #rY1)) / #rTp ** 5;

```

Symbol	Address	Type	Comment
#F_QuinticMaxAcc		Real	
#rTa		Real	
#rTp		Real	Trajectory time.
#rY0		Real	Starting point.
#rY1		Real	Ending point.

F_QuinticMaxVel [FC1]

F_QuinticMaxVel Properties					
General					
Name	F_QuinticMaxVel	**Number**	1	**Type**	FC
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Super-vision	Comment
▼ Input					
rY0	Real				Starting point.
rY1	Real				Ending point.
rTp	Real				Trajectory time.
Output					
InOut					
▼ Temp					
rT	Real	0.0			Current time
Constant					
▼ Return					
F_QuinticMaxVel	Real				

```
#rT := #rTp / 2.0;
#F_QuinticMaxVel := (60 * #rT ** 3 * (#rY0 - #rY1)) / #rTp ** 4 - (30 * #rT **
    2 * (#rY0 - #rY1)) / #rTp ** 3 - (30 * #rT ** 4 * (#rY0 - #rY1)) / #rTp ** 5;

```

Symbol	Address	Type	Comment
#F_QuinticMaxVel		Real	
#rT		Real	Current time
#rTp		Real	Trajectory time.
#rY0		Real	Starting point.
#rY1		Real	Ending point.

F_SetState [FC5]

F_SetState Properties					
General					
Name	F_SetState	**Number**	5	**Type**	FC
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Super-vision	Comment
▼ Input					
nState	Int				
Output					
InOut					
Temp					
Constant					
▼ Return					
F_SetState	Void				

```
// This function controls logic that happens once during state change

// Update state
"fbStateMachine".nCurrentState := #nState;

// Set flags to default
"fbStateMachine".bSystemOn := TRUE;
"fbStateMachine".bSystemIdle := FALSE;
"fbStateMachine".bSystemManual := FALSE;
"fbStateMachine".bSystemAutomatic := FALSE;
"fbStateMachine".bSystemService := FALSE;

// Check state and do logic
CASE #nState OF
  0: // OFF
    "fbStateMachine".sInfoMsg := 'Changed state to OFF';
    "fbStateMachine".bSystemOn := FALSE;
    "fbStateMachine".bHeaveCompActive := FALSE;

  1: // IDLE
    "fbStateMachine".sInfoMsg := 'Changed state to IDLE';
    "fbStateMachine".bSystemIdle := TRUE;

  2: // MANUAL
    "fbStateMachine".sInfoMsg := 'Changed state to MANUAL';
    "fbStateMachine".bSystemManual := TRUE;

  3: // AUTOMATIC
    "fbStateMachine".sInfoMsg := 'Changed state to AUTOMATIC';
    "fbStateMachine".bSystemAutomatic := TRUE;

    // Initialize trajector data
    "fbQuintic".rTp := "fbStateMachine".rTrajectoryTp;
    "fbQuintic".rY0 := "DataInput".rPayloadY;
```

```
    "fbQuintic".rY1 := "fbStateMachine".rTrajectoryYEnd;
    "fbStateMachine".bTrajectoryCompleted := FALSE;
    "fbStateMachine".bTrajectoryCanceled := FALSE;

  4: // SERVICE
    "fbStateMachine".sInfoMsg := 'Changed state to SERVICE';
    "fbStateMachine".bSystemService := TRUE;
END_CASE;
```

Symbol	Address	Type	Comment
"DataInput".rPayloadY	%DB301.DBD0	Real	
"fbQuintic".rTp	%DB10.DBD18	Real	
"fbQuintic".rY0	%DB10.DBD22	Real	
"fbQuintic".rY1	%DB10.DBD26	Real	
"fbStateMachine".bHeaveCompActive	%DB2.DBX10.7	Bool	True if heave compensation is activated.
"fbStateMachine".bSystemAutomatic	%DB2.DBX10.5	Bool	System automatic flag.
"fbStateMachine".bSystemIdle	%DB2.DBX10.3	Bool	System idle flag
"fbStateMachine".bSystemManual	%DB2.DBX10.4	Bool	System manual flag.
"fbStateMachine".bSystemOn	%DB2.DBX10.2	Bool	System on flag.
"fbStateMachine".bSystemService	%DB2.DBX10.6	Bool	System service flag.
"fbStateMachine".bTrajectoryCanceled	%DB2.DBX32.1	Bool	True if trajectory is canceled.
"fbStateMachine".bTrajectoryCompleted	%DB2.DBX32.0	Bool	True when trajectory is completed.
"fbStateMachine".nCurrentState	%DB2.DBW12	Int	State machine current state.
"fbStateMachine".rTrajectoryTp	%DB2.DBD24	Real	Trajectory time to complete.
"fbStateMachine".rTrajectoryYEnd	%DB2.DBD20	Real	Trajectory end position.
"fbStateMachine".sInfoMsg	P#DB2.DBX70.0	String	Info msg for hmi.
#nState		Int	

F_Sign [FC2]

F_Sign Properties					
General					
Name	F_Sign	**Number**	2	**Type**	FC
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Super-vision	Comment
▼ Input					
rValue	Real				Input value which sign should be returned.
Output					
InOut					
Temp					
Constant					
▼ Return					
F_Sign	Real				Signed value.

```
0001 #F_Sign := #rValue / ABS(#rValue);
```

Symbol	Address	Type	Comment
#F_Sign		Real	Signed value.
#rValue		Real	Input value which sign should be returned.

FB_LowPassFilter [FB6]

FB_LowPassFilter Properties					
General					
Name	FB_LowPassFilter	**Number**	6	**Type**	FB
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input									
rCutoffFrequency	Real	0.0	100.0	True	True	True	False		Break frequency.
rDt	Real	4.0	1.0	True	True	True	False		Cycle time in seconds.
rX	Real	8.0	0.0	True	True	True	False		Input value.
▼ Output									
rY	Real	12.0	0.0	True	True	True	False		Output value.
InOut									
▼ Static									
rYLast	Real	16.0	0.0	True	True	True	False		Last output value.
rOmega	Real	20.0	0.0	True	True	True	False		Angular frequency.
▼ Temp									
rAlpha	Real	0.0							Temporary coeffisient.
▼ Constant									
rPi	Real		3.141593						Constant pi.

```
// Calculate omega
#rOmega := 2 * #rPi * #rCutoffFrequency;

// Calculate dynamic constant
#rAlpha := (#rOmega*#rDt)/(1 + #rOmega*#rDt);

// Calculate output
#rY := #rAlpha*#rX + (1 - #rAlpha)*#rYLast;

// Save output as last output
#rYLast := #rY;
```

Symbol	Address	Type	Comment
#rAlpha		Real	Temporary coeffisient.
#rCutoffFrequency		Real	Break frequency.

Symbol	Address	Type	Comment
#rDt		Real	Cycle time in seconds.
#rOmega		Real	Angular frequency.
#rPi	3.141593	Real	Constant pi.
#rX		Real	Input value.
#rY		Real	Output value.
#rYLast		Real	Last output value.

FB_PID [FB9]

FB_PID Properties					
General					
Name	FB_PID	**Number**	9	**Type**	FB
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input									
COM_RST	Bool	0.0	FALSE	True	True	True	False		Reset.
P_SEL	Bool	0.1	TRUE	True	True	True	False		Use proportional term.
I_SEL	Bool	0.2	FALSE	True	True	True	False		Use integral term.
D_SEL	Bool	0.3	FALSE	True	True	True	False		Use derivative term.
Ts	Time	2.0	T#1s	True	True	True	False		Integration time.
SP_INT	Real	6.0	0.0	True	True	True	False		Internal setpoint.
PV_IN	Real	10.0	0.0	True	True	True	False		Process value input.
GAIN	Real	14.0	0.0	True	True	True	False		Proportional gain.
TI	Time	18.0	T#1s	True	True	True	False		Integration time.
TD	Time	22.0	T#1s	True	True	True	False		
CLMP	Bool	26.0	FALSE	True	True	True	False		Clamp integrator.
LMN_HLM	Real	28.0	1.0	True	True	True	False		
LMN_LLM	Real	32.0	-1.0	True	True	True	False		
DISV	Real	36.0	0.0	True	True	True	False		
▼ Output									
LMN	Real	40.0	0.0	True	True	True	False		
InOut									
▼ Static									
bInitDone	Bool	44.0	FALSE	True	True	True	False		
▼ Temp									

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
rError	Real	0.0							
rErrorIntegral	Real	4.0							
rErrorLast	Real	8.0							
rTi	Real	12.0							
rTd	Real	16.0							
rTs	Real	20.0							
rLMN	Real	24.0							
rLMN_P	Real	28.0							
rLMN_I	Real	32.0							
rLMN_D	Real	36.0							
Constant									

```
// Reset controller
IF #COM_RST OR NOT #bInitDone THEN
  #rError := 0.0;
  #rErrorIntegral := 0.0;
  #rErrorLast := 0.0;
  #rTi := 0.0;
  #rTs := 0.0;
  #rLMN_P := 0.0;
  #rLMN_I := 0.0;
  #rLMN_D := 0.0;
  #bInitDone := TRUE;
END_IF;

// Convert time-step from time to real
#rTs := DINT_TO_REAL(TIME_TO_DINT(#Ts)) * (10 ** -3);

// Calculate error
#rError := #SP_INT - #PV_IN;

// Calculate proportional contribution
IF #P_SEL THEN
  #rLMN_P := #GAIN * #rError;
ELSE
  #rLMN_P := 0.0;
END_IF;

// Calculate integral contribution
IF #I_SEL THEN
  #rTi := DINT_TO_REAL(TIME_TO_DINT(#TI)) * (10 ** -3);
  #rErrorIntegral := #rErrorIntegral + (#rError * #rTs); // integrate error
  #rLMN_I := (#GAIN / #rTi) * #rErrorIntegral;

  IF #CLMP THEN
    IF #rLMN_I > #LMN_HLM THEN
      #rLMN_I := #LMN_HLM;
      #rErrorIntegral := #rErrorIntegral - #rError * #rTs;
    ELSIF #rLMN_I < #LMN_LLM THEN
      #rLMN_I := #LMN_LLM;
```

```
      #rErrorIntegral := #rErrorIntegral - #rError * #rTs;
    END_IF;
  END_IF;
ELSE
  #rLMN_I := 0.0;
END_IF;

// Calculate derivative contribution
IF #D_SEL THEN
  #rTd := DINT_TO_REAL(TIME_TO_DINT(#TD)) * (10 ** -3);
  #rLMN_D := (#GAIN * #rTd) * ((#rError - #rErrorLast) / #rTs);
  #rErrorLast := #rError;
ELSE
  #rLMN_D := 0.0;
END_IF;

// Calculate total output
#rLMN := #rLMN_P + #rLMN_I + #rLMN_D;

// Saturate output
IF #rLMN > #LMN_HLM THEN
  #rLMN := #LMN_HLM;
ELSIF #rLMN < #LMN_LLM THEN
  #rLMN := #LMN_LLM;
END_IF;

// Add feedforward
#LMN := #rLMN + #DISV;
```

Symbol	Address	Type	Comment
#bInitDone		Bool	
#CLMP		Bool	Clamp integrator.
#COM_RST		Bool	Reset.
#D_SEL		Bool	Use derivative term.
#DISV		Real	
#GAIN		Real	Proportional gain.
#I_SEL		Bool	Use integral term.
#LMN		Real	
#LMN_HLM		Real	
#LMN_LLM		Real	
#P_SEL		Bool	Use proportional term.
#PV_IN		Real	Process value input.
#rError		Real	
#rErrorIntegral		Real	
#rErrorLast		Real	
#rLMN		Real	
#rLMN_D		Real	
#rLMN_I		Real	
#rLMN_P		Real	
#rTd		Real	
#rTi		Real	
#rTs		Real	
#SP_INT		Real	Internal setpoint.
#TD		Time	
#TI		Time	Integration time.
#Ts		Time	Integration time.

FB_Quintic [FB4]

FB_Quintic Properties					
General					
Name	FB_Quintic	**Number**	4	**Type**	FB
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input									
rLastCycleTime	Real	0.0	0.0	True	True	True	False		Last cycle time.
▼ Output									
rYRef	Real	4.0	0.0	True	True	True	False		
rYdRef	Real	8.0	0.0	True	True	True	False		
bTrajectoryCompleted	Bool	12.0	false	True	True	True	False		
InOut									
▼ Static									
rT	Real	14.0	0.0	True	True	True	False		
rTp	Real	18.0	0.0	True	False	False	False		
rY0	Real	22.0	0.0	True	False	False	False		
rY1	Real	26.0	0.0	True	False	False	False		
bReset	Bool	30.0	false	True	True	True	False		
Temp									
▼ Constant									
rYTolreance	Real		0.01						Tolerance for when trajectory is complete.

```
// Check if reset is called
IF #bReset THEN
  #rT := 0.0;
  #bTrajectoryCompleted := FALSE;
  #bReset := FALSE;
END_IF;

// Iterate and saturate time
IF #rT < #rTp THEN
```

```
  #rT := #rT + #rLastCycleTime;
ELSE
  #bTrajectoryCompleted := TRUE;
  #rT := #rTp;
END_IF;

// Calculate outputs
#rYRef := #rY0 - (10 * #rT ** 3 * (#rY0 - #rY1)) / #rTp ** 3 + (15 * #rT ** 4 * (#rY0 - #rY1)) / #rTp ** 4 - (6 * #rT ** 5 * (#rY0 - #rY1)) / #rTp ** 5;
#rYdRef := (60 * #rT ** 3 * (#rY0 - #rY1)) / #rTp ** 4 - (30 * #rT ** 2 * (#rY0 - #rY1)) / #rTp ** 3 - (30 * #rT ** 4 * (#rY0 - #rY1)) / #rTp ** 5;
// rYdd := (180*t^2*(y0 - y1))/t1^4 - (60*t*(y0 - y1))/t1^3 - (120*t^3*(y0 - y1))/t1^5

IF ABS(#rYRef - #rY1) < #rYTolreance THEN
  #bTrajectoryCompleted := TRUE;
  #bReset := TRUE;
  #rYRef := #rY1;
END_IF;

```

Symbol	Address	Type	Comment
#bReset		Bool	
#bTrajectoryCompleted		Bool	
#rLastCycleTime		Real	Last cycle time.
#rT		Real	
#rTp		Real	
#rY0		Real	
#rY1		Real	
#rYdRef		Real	
#rYRef		Real	
#rYTolreance	0.01	Real	Tolerance for when trajectory is complete.

FB_Ramp [FB7]

FB_Ramp Properties					
General					
Name	FB_Ramp	**Number**	7	**Type**	FB
Language	SCL	**Numbering**	Automatic		
Information					
Title		**Author**		**Comment**	
Family		**Version**	0.1	**User-defined ID**	

Name	Data type	Offset	Default value	Accessible from HMI/OPC UA	Writable from HMI/OPC UA	Visible in HMI engineering	Setpoint	Supervision	Comment
▼ Input									
rEndValue	Real	0.0	0.0	True	True	True	False		Desired end value.
rRampTime	Real	4.0	0.0	True	True	True	False		Desired ramp time.
rCurrentValue	Real	8.0	0.0	True	True	True	False		Current value.
rLastCycleTime	Real	12.0	0.0	True	True	True	False		Last cycle time (dt).
Output									
▼ InOut									
rReference	Real	16.0	0.0	True	True	True	False		Reference inout.
▼ Static									
rSlope	Real	20.0	0.0	True	True	True	False		Ramp (dy/dt).
rEndValueMem	Real	24.0	0.0	True	True	True	False		End value memory.
rTolerance	Real	28.0	0.1	True	True	True	False		Preliminary tolerance.
bTrajectoryCompleted	Bool	32.0	false	True	True	True	False		
Temp									
Constant									

```
// Check if end value has changed
IF #rEndValueMem <> #rEndValue THEN
  #rSlope := (#rEndValue - #rCurrentValue) / #rRampTime;
  #rEndValueMem := #rEndValue;
  #bTrajectoryCompleted := FALSE;
END_IF;

// Ramp
IF ABS(#rEndValue - #rCurrentValue) <= #rTolerance THEN
  #bTrajectoryCompleted := TRUE;
END_IF;

```

```
IF #bTrajectoryCompleted THEN
  #rReference := #rEndValue;
ELSE
  #rReference := #rReference + #rSlope * #rLastCycleTime;
END_IF;
```

Symbol	Address	Type	Comment
#bTrajectoryCompleted		Bool	
#rCurrentValue		Real	Current value.
#rEndValue		Real	Desired end value.
#rEndValueMem		Real	End value memory.
#rLastCycleTime		Real	Last cycle time (dt).
#rRampTime		Real	Desired ramp time.
#rReference		Real	Reference inout.
#rSlope		Real	Ramp (dy/dt).
#rTolerance		Real	Preliminary tolerance.

G Catalogs

G.1 Motor Catalog Data

G.2 Drive Catalog Data

Flameproof cast iron motors
Technical data for Ex d IIB/IIC T4 Gb

IP 55 - IC 411 - Insulation class F, temperature rise class B
IE2 efficiency class according to IEC 60034-30; 2008

Output kW	Motor type	Product code	Speed r/min	Efficiency IEC 60034--2-1; 2007 Full load 100%	3/4 load 75%	1/2 load 50%	Power factor cos φ	Current I_N A	I_s / I_N	Torque T_N Nm	T_l / T_N	T_b / T_N	Moment of inertia J = 1/4 GD^2 kgm²	Weight kg	Sound pressure level L_{PA} dB
1500 r/min = 4-poles		**400 V 50 Hz**						**CENELEC-design**							
0.55	**M3JP 80 MA**	3GJP 082 310-••H	1421	76.6	76.6	73.7	0.73	1.41	4.9	3.6	2.3	2.7	0.001	38	59
0.75	**M3JP 80 MB**	3GJP 082 320-••H	1412	80.4	80.5	78.4	0.76	1.77	5.2	5	2.2	2.7	0.0012	40	59
1.1	**M3JP 90 SLA**	3GJP 092 010-••H	1432	83.3	83.3	80.7	0.77	2.4	5.9	7.3	2.8	3.5	0.002	51	54
1.5	**M3JP 90 SLC**	3GJP 092 030-••H	1431	83.2	82.8	80.4	0.79	3.2	6.5	10	2.3	3.0	0.003	53	54
2.2	**M3JP 100 LA**	3GJP 102 510-••H	1441	84.7	85.6	84.8	0.86	4.3	7.0	14.5	2.7	3.3	0.0075	70	52
3	**M3JP 100 LB**	3GJP 102 520-••H	1442	86.5	87.2	86.3	0.83	6	7.3	19.8	2.7	3.4	0.0081	72	52
4	**M3JP 112 MC**	3GJP 112 330-••H	1458	88.2	87.8	85.6	0.78	8.3	8.7	26.1	3.0	3.8	0.013	81	52
5.5	**M3JP 132 SMB**	3GJP 132 220-••H	1458	88.5	88.7	87.2	0.79	11.3	7.4	36	3.0	3.5	0.023	111	60
7.5	**M3JP 132 SMD**	3GJP 132 240-••H	1460	89.1	89.1	87.6	0.75	16.1	6.8	49	3.3	3.7	0.034	114	60
11	**M3JP 160 MLC**	3GJP 162 430-••H	1470	91.2	91.5	90.6	0.82	21.2	7.8	71.4	3.0	3.5	0.096	232	62
15	**M3JP 160 MLE**	3GJP 162 450-••H	1467	92.0	92.4	92.1	0.84	28	7.8	97.6	3.0	3.5	0.13	255	61
18.5	**M3JP 180 MLA**	3GJP 182 410-••H	1474	91.6	92.0	91.2	0.83	35.1	7.2	119	2.6	3.1	0.19	277	62
22	**M3JP 180 MLB**	3GJP 182 420-••H	1471	91.6	92.4	92.2	0.83	41.7	6.8	142	2.5	3.0	0.21	285	62
30	**M3JP 200 MLB**	3GJP 202 420-••G	1475	93.6	94.0	93.7	0.85	54.4	7.4	194	3.0	2.8	0.34	340	61
37	**M3JP 225 SMB**	3GJP 222 220-••G	1480	93.6	93.9	93.4	0.85	67.1	7.6	238	3.2	2.9	0.42	390	67
45	**M3JP 225 SMC**	3GJP 222 230-••G	1477	94.1	94.6	94.4	0.88	78.4	7.6	290	3.2	2.7	0.49	425	67
55	**M3JP 250 SMA**	3GJP 252 210-••G	1479	94.3	94.3	93.6	0.84	100	7.2	355	2.5	3.1	0.72	450	66
75	**M3JP 280 SMA**	3GJP 282 210-••G	1484	94.5	94.5	93.9	0.85	134	6.9	482	2.5	2.8	1.25	725	68
90	**M3JP 280 SMB**	3GJP 282 220-••G	1483	94.7	94.8	94.4	0.86	159	7.2	579	2.5	2.7	1.5	765	68
110	**M3JP 315 SMA**	3GJP 312 210-••G	1487	95.1	95.1	94.3	0.86	194	7.2	706	2.0	2.5	2.3	1000	70
132	**M3JP 315 SMB**	3GJP 312 220-••G	1487	95.4	95.4	94.7	0.86	232	7.1	847	2.3	2.7	2.6	1060	70
160	**M3JP 315 SMC**	3GJP 312 230-••G	1487	95.6	95.6	95.1	0.85	284	7.2	1027	2.4	2.9	2.9	1100	70
200	**M3JP 315 MLA**	3GJP 312 410-••G	1486	95.6	95.6	95.3	0.86	351	7.2	1285	2.5	2.9	3.5	1260	70
250	**M3JP 355 SMA**	3GJP 352 210-••G	1488	95.9	95.9	95.5	0.86	437	7.1	1604	2.3	2.7	5.9	1800	74
315	**M3JP 355 SMB**	3GJP 352 220-••G	1488	95.9	95.9	95.6	0.86	551	7.3	2021	2.3	2.8	6.9	1970	74
355	**M3JP 355 SMC**	3GJP 352 230-••G	1487	95.9	95.9	95.7	0.86	621	6.8	2279	2.4	2.7	7.2	2010	78
400	**M3JP 355 MLA**	3GJP 352 410-••G	1489	96.3	96.3	95.9	0.85	705	6.8	2565	2.3	2.6	8.4	2330	78
450	**M3JP 355 MLB**	3GJP 352 420-••G	1490	96.8	96.8	96.3	0.86	780	6.9	2884	2.3	2.9	8.4	2330	78
500	**M3JP 355 LKA**	3GJP 352 810-••G	1490	97.0	97.0	96.5	0.86	865	6.8	3204	2.0	3.0	10	2690	78
560	**M3JP 400 LA**	3GJP 402 510-••G	1491	96.8	96.8	96.3	0.85	982	7.4	3586	2.4	2.8	15	3200	78
560	**M3JP 400 LKA**	3GJP 402 810-••G	1491	96.8	96.8	96.3	0.85	982	7.4	3586	2.4	2.8	15	3200	78
630	**M3JP 400 LB**	3GJP 402 520-••G	1491	97.0	97.0	96.5	0.87	1077	7.6	4034	2.2	2.9	16	3580	78
630	**M3JP 400 LKB**	3GJP 402 820-••G	1491	97.0	97.0	96.5	0.87	1077	7.6	4034	2.2	2.9	16	3580	78
710 [1]	**M3JP 400 LC**	3GJP 402 530-••G	1491	97.1	97.1	96.6	0.86	1227	7.6	4547	2.4	3.0	17	3680	78
710 [1]	**M3JP 400 LKC**	3GJP 402 830-••G	1491	97.1	97.1	96.6	0.86	1227	7.6	4547	2.4	3.0	17	3680	78
1500 r/min = 4-poles		**400 V 50 Hz**						**High-output design**							
18.5	**M3JP 160 MLF**	3GJP 162 460-••H	1469	91.7	92.1	91.4	0.83	35	7.8	120	3.2	3.5	0.13	255	68
22 [2]	**M3JP 160 MLG**	3GJP 162 470-••H	1466	90.8	91.1	90.4	0.81	43.1	7.9	143	3.3	3.6	0.13	255	68
30 [1,2]	**M3JP 180 MLC**	3GJP 182 430-••H	1473	92.2	92.3	91.6	0.81	57.9	7.1	194	2.8	3.2	0.248	304	66
37	**M3JP 200 MLC**	3GJP 202 430-••G	1475	93.0	93.1	92.3	0.82	70	7.5	239	3.5	3.2	0.34	340	73
55	**M3JP 225 SMD**	3GJP 222 240-••G	1483	94.3	94.5	93.9	0.83	101	7.4	354	3.4	2.9	0.55	445	68
62 [2]	**M3JP 225 SME**	3GJP 222 250-••G	1477	93.5	93.7	93.0	0.84	113	7.7	400	3.5	2.9	0.55	445	74
75	**M3JP 250 SMB**	3GJP 252 220-••G	1476	94.3	94.5	94.2	0.86	133	7.6	485	2.8	3.2	0.88	505	73
86 [2]	**M3JP 250 SMC**	3GJP 252 230-••G	1477	94.1	94.4	94.0	0.85	155	7.8	556	2.9	3.5	0.98	530	74
110	**M3JP 280 SMC**	3GJP 282 230-••G	1485	95.1	95.2	94.7	0.86	194	7.6	707	3.0	3.0	1.85	825	68

[1] Temperature rise class F
[2] Efficiency class IE1

The two bullets in the product code indicate choice of mounting arrangements, voltage and frequency code (see ordering information page).

I_s / I_N = Starting current
T_l / T_N = Locked rotor torque
T_b / T_N = Pull-out torque

Efficiency values are given according to IEC 60034-2-1; 2007. Please note that the values are not comparable without knowing the testing method.
ABB has calculated the efficiency values according to indirect method, stray load losses (additional losses) determined from measuring.

Ratings, types and voltages
Wall-mounted drives, ACS880-01

U_N = 230 V (range 208 to 240 V). The power ratings are valid at nominal voltage 230 V (0.55 to 75 kW).

Nominal ratings			Light-overload use		Heavy-duty use		Noise level	Heat dissipation	Air flow	Type designation	Frame size
I_N A	I_{max} A	P_N kW	I_{Ld} A	P_{Ld} kW	I_{Hd} A	P_{Hd} kW	dBA	W	m^3/h		
4.6	6.3	0.75	4.4	0.75	3.7	0.55	46	73	44	ACS880-01-04A6-2	R1
6.6	7.8	1.1	6.3	1.1	4.6	0.75	46	94	44	ACS880-01-06A6-2	R1
7.5	11.2	1.5	7.1	1.5	6.6	1.1	46	122	44	ACS880-01-07A5-2	R1
10.6	12.8	2.2	10.1	2.2	7.5	1.5	46	172	44	ACS880-01-10A6-2	R1
16.8	18.0	4.0	16.0	4.0	10.6	2.2	51	232	88	ACS880-01-16A8-2	R2
24.3	28.6	5.5	23.1	5.5	16.8	4	51	337	88	ACS880-01-24A3-2	R2
31.0	41	7.5	29.3	7.5	24.3	5.5	57	457	134	ACS880-01-031A-2	R3
46	64	11	44	11	38	7.5	62	500	200	ACS880-01-046A-2	R4
61	76	15	58	15	45	11	62	630	200	ACS880-01-061A-2	R4
75	104	18.5	71	18.5	61	15	62	680	280	ACS880-01-075A-2	R5
87	122	22	83	22	72	18.5	62	730	280	ACS880-01-087A-2	R5
115	148	30	109	30	87	22	67	840	435	ACS880-01-115A-2	R6
145	178	37	138	37	105	30	67	940	435	ACS880-01-145A-2	R6
170	247	45	162	45	145	37	67	1260	450	ACS880-01-170A-2	R7
206	287	55	196	55	169	45	67	1500	450	ACS880-01-206A-2	R7
274	362	75	260	75	213	55	65	2100	550	ACS880-01-274A-2	R8 [3]

U_N = 400 V (range 380 to 415 V). The power ratings are valid at nominal voltage 400 V (0.55 to 250 kW).

Nominal ratings			Light-overload use		Heavy-duty use		Noise level	Heat dissipation	Air flow	Type designation	Frame size
I_N A	I_{max} A	P_N kW	I_{Ld} A	P_{Ld} kW	I_{Hd} A	P_{Hd} kW	dBA	W	m^3/h		
2.4	3.1	0.75	2.3	0.75	1.8	0.55	46	30	44	ACS880-01-02A4-3	R1
3.3	4.1	1.1	3.1	1.1	2.4	0.75	46	40	44	ACS880-01-03A3-3	R1
4.0	5.6	1.5	3.8	1.5	3.3	1.1	46	52	44	ACS880-01-04A0-3	R1
5.6	6.8	2.2	5.3	2.2	4.0	1.5	46	73	44	ACS880-01-05A6-3	R1
8	9.5	3.0	7.6	3.0	5.6	2.2	46	94	44	ACS880-01-07A2-3	R1
10	12.2	4.0	9.5	4.0	8	3	46	122	44	ACS880-01-09A4-3	R1
12.9	16.0	5.5	12.0	5.5	10	4	46	172	44	ACS880-01-12A6-3	R1
17	21	7.5	16	7.5	12.6	5.5	51	232	88	ACS880-01-017A-3	R2
25	29	11	24	11	17	7.5	51	337	88	ACS880-01-025A-3	R2
32	42	15	30	15	25	11	57	457	134	ACS880-01-032A-3	R3
38	54	18.5	36	18.5	32	15	57	562	134	ACS880-01-038A-3	R3
45	64	22	43	22	38	18.5	62	667	200	ACS880-01-045A-3	R4
61	76	30	58	30	45	22	62	907	200	ACS880-01-061A-3	R4
72	104	37	68	37	61	30	62	1117	280	ACS880-01-072A-3	R5
87	122	45	83	45	72	37	62	1120	280	ACS880-01-087A-3	R5
105	148	55	100	55	87	45	67	1295	435	ACS880-01-105A-3	R6
145	178	75	138	75	105	55	67	1440	435	ACS880-01-145A-3	R6
169	247	90	161	90	145	75	67	1940	450	ACS880-01-169A-3	R7
206	287	110	196	110	169	90	67	2310	450	ACS880-01-206A-3	R7
246	350	132	234	132	206	110	65	3300	550	ACS880-01-246A-3	R8
293	418	160	278	160	246 [1]	132	65	3900	550	ACS880-01-293A-3	R8 [3]
363	498	200	345	200	293	160	68	4800	1150	ACS880-01-363A-3	R9 [6]
430	545	250	400	200	363 [2]	200	68	6000	1150	ACS880-01-430A-3	R9 [5]

U_N = 500 V (range 380 to 500 V). The power ratings are valid at nominal voltage 500 V (0.55 to 250 kW).

Nominal ratings			Light-overload use		Heavy-duty use		Noise level	Heat dissipation	Air flow	Type designation	Frame size
I_N A	I_{max} A	P_N kW	I_{Ld} A	P_{Ld} kW	I_{Hd} A	P_{Hd} kW	dBA	W	m^3/h		
2.1	3.1	0.75	2.0	0.75	1.7	0.55	46	30	44	ACS880-01-02A1-5	R1
3.0	4.1	1.1	2.8	1.1	2.1	0.75	46	40	44	ACS880-01-03A0-5	R1
3.4	5.6	1.5	3.2	1.5	3.0	1.1	46	52	44	ACS880-01-03A4-5	R1
4.8	6.8	2.2	4.6	2.2	3.4	1.5	46	73	44	ACS880-01-04A8-5	R1
5.2	9.5	3.0	4.9	3.0	4.8	2.2	46	94	44	ACS880-01-05A2-5	R1
7.6	12.2	4.0	7.2	4.0	5.2	3	46	122	44	ACS880-01-07A6-5	R1
11.0	16.0	5.5	10.4	5.5	7.6	4	46	172	44	ACS880-01-11A0-5	R1
14	21	7.5	13	7.5	11	5.5	51	232	88	ACS880-01-014A-5	R2
21	29	11	19	11	14	7.5	51	337	88	ACS880-01-021A-5	R2
27	42	15	26	15	21	11	57	457	134	ACS880-01-027A-5	R3
34	54	18.5	32	18.5	27	15	57	562	134	ACS880-01-034A-5	R3
40	64	22	38	22	34	19	62	667	200	ACS880-01-040A-5	R4
52	76	30	49	30	40	22	62	907	200	ACS880-01-052A-5	R4
65	104	37	62	37	52	30	62	1117	280	ACS880-01-065A-5	R5
77	122	45	73	45	65	37	62	1120	280	ACS880-01-077A-5	R5
96	148	55	91	55	77	45	67	1295	435	ACS880-01-096A-5	R6
124	178	75	118	75	96	55	67	1440	435	ACS880-01-124A-5	R6
156	247	90	148	90	124	75	67	1940	450	ACS880-01-156A-5	R7
180	287	110	171	110	156	90	67	2310	450	ACS880-01-180A-5	R7
240	350	132	228	132	180	110	65	3300	550	ACS880-01-240A-5	R8 [4]
260	418	160	247	160	240 [1]	132	65	3900	550	ACS880-01-260A-5	R8 [3]
361	542	200	343	200	302	200	68	4800	1150	ACS880-01-361A-5	R9 [6]
414	542	250	393	250	361 [2]	200	68	6000	1150	ACS880-01-414A-5	R9 [5]