Automatic Procedure for Determining Control Parameters for Ship-to-Shore Cranes

Hampus Hellström

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT- 6071
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Printed in Sweden by Tryckeriet i E-huset
Lund 2017

Abstract

The objective of this thesis has been to create a tuning functionality that can determine the control parameters for a crane that loads and unloads containers from ships. To accomplish this a model of the crane has been developed in MapleSIM and exported into MATLAB's Simulink. In MATLAB cost functions have been developed and evaluated to later be used in optimisation algorithms to find the optimal parameters.

There are two sets of control parameters that the script needs to determine, the first is composed of 6 subsets of 4 co-dependent parameters that need to be optimised together. The other set consists of 66 parameters that are independent of each other. A few different optimisation methods have been considered, but for the first set a particle swarm optimisation was used and for the second set, due to the cost function, a form of binary search was possible to use.

As concluding results, we found that the underlying model of the crane was not accurate enough to determine the first set of control parameters. The second set of control parameters are similar enough to the real values and can possibly be used on a real crane.

Acknowledgement

I would like to direct my thanks to my supervisor Jonas Öhr at ABB Crane system where the thesis work was conducted who have helped me through this project. I would also like to extend my thanks to the MapleSIM support team in Cambridge for their help solving any MapleSIM related problems that arose during the thesis.

Lastly I would like to direct my thanks to my supervisor Anders Robertsson at the Faculty of Engineering, Lund University for his patience and his help.

Contents

1 Introduction

1.1 Background

Automation has become more prevalent in all industries and since every controller needs a set of parameters to ensure it operates as efficiently as possible for its set of tasks, therefore, efficient ways of determining these control parameters are needed so not to spend resources manually testing for a good set of parameters.

ABB Cranes [2] develops Electronic Load Controller (ELC) for cranes that move containers between freight ship and the harbour, so called Ship-To-Shore cranes (STS), see Figure 1. The purpose of the ELC is to either follow a predetermined path or follow a desired velocity while dampening the swaying in the load.

ABB currently has a controller in the ELC but determining the control parameters is a time-consuming task that requires a specialist on site, and valuable time that could be more efficiently used and needed elsewhere. In order to minimise this time, a method for determining these parameters before the specialist arrives is needed.

Figure 1: STS cranes loading containers onto fright ships

1.2 Thesis purpose

The purpose of this thesis is to extend the existing MapleSIM model for an STS crane. The current MapleSIM model does neither take into account the elasticity of the cables driving the trolley nor the effects of the hydraulics that make sure the tension in the cables are not too high or too low. Furthermore, this model needs to be general enough so that by changing the physical parameters it should be able to simulate any crane.

This model should later be exported into Simulink and combined with the existing controllers used in the actual cranes. A suitable optimisation algorithm

needs to be implemented in order to determine the control parameters. There are two sets of a total of 90 different parameter that need to be determined. The first set makes up 24 out of the 90 parameters,X_i, Y_i, Z_i and W_i, $i \in [1, 6]$, where each index corresponds to one subset of parameters used at a specific region of the workspace, so-called gain-scheduling [5]. What subset should be used, or what i is, depends on how long the cables are that the container is hanging from. Longer cables result in a longer pendulum which needs different parameters. These parameters are used to stabilise the container when a human is steering by controlling the velocity of the container. Therefore the scenario these parameters will be optimised for is when the crane operator releases the joystick and the reference velocity goes to zero. The parameters stabilising the container from swaying should do so in a manner that is both efficient and user friendly, that is, it behaves as the operators expect it to. As the optimal values for X_i, Y_i, Z_i and $W_i$all depend on each other they will have to be optimised together, resulting in six 4-dimensional optimisation problems.

The other 66 parameters, the position control gains, $P_c^{I_{rl},I_d}$ can be optimised one by one, resulting in another 66 1-dimensional optimisation problems. These parameters are used when the container is set to move to a specific location, and therefore no driver will be involved. Thus these will be optimised for minimising the time it will take for the container to stabilise at the desired position, given a certain distance it will be required to move. Similar to the parameters X_i, Y_i, Z_i and W_i, each $P_c^{I_{rl},I_d}$ has an index I_{rl} defined for the same above regions as mentioned above. In addition this index is $I_d \in [1, 11]$ indicates which of a fixed set of distances the container needs to move between the initial location to the desired location, is used. As there are a lot of parameters that need to be determined and each simulation will have to be tested for multiple different container masses an efficient optimisation algorithm is needed to be able to complete this task in a timely manner.

1.3 Simulation Environments

The main programs that will be used in this thesis are MATLAB with Simulink and MapleSIM. As part of the model had already had been built in MapleSIM it was a natural choice to continue in this modelling environment. Since MATLAB and Simulink are used at ABB and MapleSIM can export models as Simulink-blocks, therefore running the optimisation in MATLAB and Simulink seemed to be the best choice.

1.4 Confidentiality

Since parts of this report will be confidential, some parts may be lacking some information. However when this is the case the reader will be noted and the time axis on plots will be normalised to the maximum value of the plot.

2 Theory

2.1 MapleSIM

MapleSIM is a product of Maplesoft [7]. It is a Modelica-based modelling and simulation program. The program runs the symbolic mathematical engine of Maple, a symbolic mathematical program also developed by Maplesoft. A system of symbolic equations representing the model is generated by combining sub models and components which each has symbolic equations defined for how they behave and interact with neighbouring components. The individual components are combined via a graphical interface, thus MapleSIM allows to build complex model from simple components.
MapleSIM has built-in functions that allow the user to convert a MapleSIM model to C file and also generate a script that creates a `.m` script which in MATLAB creates a Simulink block from the C file.

2.2 MATLAB and Simulink

MATLAB and Simulink are products from MathWorks [8] which are mainly used for numerical computing and simulation. Simulink is a visual modelling tool, that can use scripts written in MATLAB and also be run from scripts in MATLAB. Together they can be combined to produce models of systems and to analyse how these system behave. Most code written in this thesis project will be written in MATLAB.

2.2.1 Parallel computing in MATLAB

Most computers today have multiple cores in their processors, allowing calculations to run simultaneously on each core. When running simulations in Simulink MATLAB regularly does not utilise this. However with the function `parpool`, MATLAB creates a pool of workers. This allows the `parfor`-loop to assign the work to separate workers and simultaneously compute the work assigned to each worker. Since the computation is running simultaneously `parfor` requires that no loop can be dependant on information from another loop. This proves to be a problem for some optimisation algorithms like Nelder-Meads Simplex algorithm which is explained in a later section, 3.2.4, of the report.

To make the parallel computation as effective as possible all workers should be used at all time if possible. Therefore dividing the work into a multiple of the number of worker would be the best. If this is not possible, the work should be divided into as small parts as possible in order to use all workers as much as possible.

2.3 Multidimensional optimisation algorithms

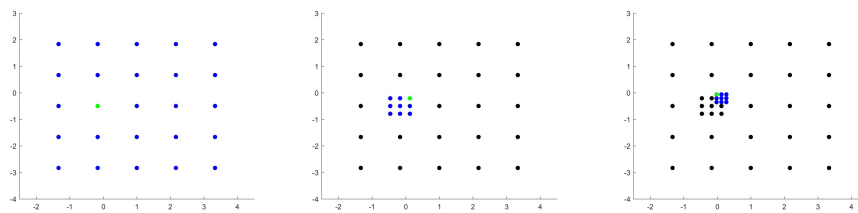
For the multidimensional optimisation, there are 6 sets of 4 dimensional optimisation. Since each simulation is rather time consuming an optimisation algorithm that calls the function as few times as possible is desired, rather than being efficiently implemented. As there are only 6 sets of optimisation that need to be run, running one optimisation on each worker would not be optimal. Most computers today have 4 core processor, which means that the last two optimisations will run on two of the workers while the other two workers will be idle thus not using the computer's full capacity. Therefore an optimisation algorithm that can utilise all cores or workers at the same time is preferable.

2.3.1 Discretisation

The first method considered is a simple discretisation method. It is a simple approach to find the minimum of a function without any knowledge of its derivative. The version tested here generates a mesh with p_i equidistant points along the i-th axis. Let the search interval along the i-th axis be between b_l^i and b_h^i. The distance between these point will then be set to $\Delta_j = (b_h^i - b_l^i)/p_j^i$, where j is the iteration number. After all points have been evaluated the point with the lowest value will be the centre point for the grid and the Δ will be updated according to:

$$\Delta_{j+1}^i = 2\frac{\Delta_j^i}{p_j^i + 1} \tag{1}$$

Thus the new grid will fit inside the n-dimensional cube around the last minimum, as depicted in Figure 3 for $n = 2$.



(a) Iteration 1. (b) Iteration 2. (c) Iteration 3.

Figure 3: The blue points are the points included in the current iterations. The green point is the one corresponding to the lowest value and so it will be the centre for the next iteration.

The disadvantage of this method is that it is slow. As the number of times the function has to be called for each grid is $\prod_{i=1}^{n} p_i$, meaning it scales exponentially with the number of dimensions. Therefore a more advance algorithm is needed; one that chooses the evaluation points in a better way. One example is the Nelder-Mead Simplex algorithm.

2.3.2 Nelder-Mead Simplex method

Nelder-Mead Simplex method (or just Simplex method) is one of the more popular method for derivative free optimisation. This method is more sophisticated than the discretisation method. Rather than making a grid of of each parameter and evaluate at each intersection the Simplex method uses a few points and from those calculating, according to certain rules, the next point where to evaluate. It starts by choosing $n + 1$ points, n being the number of dimensions. Then for each iteration it either replaces the point corresponding to the highest value of the function or all points except the point with the lowest value depending on the set of rules it works by. One version of the Nelder-Mead Simplex method, as it is described by J. A. Nelder and R. Mead in their original paper [10], is described below:

The method starts by choosing a set of $n + 1$ points, where n is the number of dimensions. The vertices are chosen to form a simplex, which is a $n + 1$-dimensional polytope. Let $f(x)$ be the function to be minimised and let x_i, $i \in \{1, 2...n, n + 1\}$ be the points for the current simplex.

Step 1: Define x_l and x_h such that:

$$f(x_l) = \min_i(f(x_i)) \tag{2}$$

$$f(x_h) = \max_i(f(x_i)) \tag{3}$$

Calculate the centroid, x_0, of all points except x_h and the reflection point, x_r, such that:

$$x_0 = \frac{1}{n} \sum_{i \neq h} x_i \tag{4}$$

$$x_r = x_0 + \alpha(x_0 - x_h) \tag{5}$$

where α is a positive constant. If $f(x_r) < f(x_l)$ move on to step 2. Else if $f(x_r) < f(x_i), \forall i \neq h$ move on to step 3. Otherwise replace x_h with x_r and restart Step 1.

Step 2: Calculate the expansion point x_e.

$$x_e = x_0 + \gamma(x_r - x_0) \tag{6}$$

where $\gamma > 0$ is a constant. If the value at the expansion point is lower than the reflection point replace x_h with x_e otherwise replace x_h with x_r and go back to step 1.

Step 3: If $f(x_r) < f(x_h)$ then replace x_h with x_r. Now calculate the contraction point, x_c:

$$x_c = x_0 + \beta(x_h - x_0) \tag{7}$$

where β is a constant such $0 < \beta \leq 0.5$. If $f(x_c) < f(x_h)$, replace x_h with x_c and return to step 1. If $f(x_c) > f(x_h)$ replace all points except x_l with

$$x_i = \frac{1}{2}(x_i + x_l), \quad \forall i \neq l \tag{8}$$

then return to step 1.

Each time before returning to step 1 check if the minimising criterion has been met and abort the algorithm if so. A flow chart over this method can be seen in appendix A.

One disadvantage of the Nelder-Meads Simplex method is that it has little possibility for parallellising as, most of the time, the next point which should be evaluated is not know until the value of the previous point has been calculated. The only time this is not true is during the initialisation and when a the simplex contracts, where it has to compute $n+1$ and n points respectively.

2.3.3 Genetic Algorithm

The Genetic Algorithm, GA from now on, tries to mimic the evolution seen in species from the process of natural selection. There are three main ways to generate a new generation; these are through the processes of selection, crossover and mutation [3]. Along with these there are other mechanisms like elitism [11].

In order to find a minimum to the function $f(x)$ the GA starts by selecting a random initial population, consisting of the individuals $b_1, b_2 ... b_n$. Each individual's performance is evaluated, and then the selection process begins.

The selection process removes the poorer performing individuals of the population. This is the main driving force behind the algorithm. The selection process is a random process where the chance of removing an individual is dependant on it performance. One simple way of selecting which individuals to remove would be by randomly selecting a certain portion of the population, where the probability of selecting an individual would increase as its fitness decreases.

$$P(b_i \; is \; removed) = \frac{f(b_i)}{\sum(f(b_i))_i} \tag{9}$$

This requires the function $f(x)$ to be positive for all x. If $f(x)$ is negative for some x, a new function needs to be introduced, $\phi(x)$. This function needs to be non-negative and non-decreasing. If $\phi(x)$ fulfils these requirements then the probability of removing an individual can be modified to instead be given by:

$$P(b_i \; is \; removed) = \frac{\phi(f(b_i))}{\sum(\phi(f(b_i)))_i} \tag{10}$$

To ensure that the cost function is always decreasing, elitism can be introduced, which will keep a certain portion of the best performing individuals in the population to next generation.

The crossover process attempts to mimic the mating process in nature; a process which combines the genetic material of two individuals that has "survived" the selection process. This process will randomly choose two individual in the remaining population and combine genetic material. The amount inherited from each parent is randomly chosen through the crossover method. The following crossover method is called Blend Crossover or $BLX-\alpha$, [6]. Through Blend Crossover, the offspring O is generated from its parents P_1 and P_2 with the following equations:

$$\begin{aligned} O &= (1-\gamma)P_1 + \gamma P_2 \\ \gamma &= (1+2\alpha)u - \alpha \end{aligned} \tag{11}$$

where u is a uniformly distributed random number between 0 and 1. α is the exploration number. Setting $\alpha = 0$ will cause the crossover method to coincide with what is called Arithmetic Crossover method, see Figure 4, while an $\alpha > 0$ would yield a larger space for where the offsprings can be generated, see Figure 5. It becomes clear why α is called the exploration number as it describes how much outside of the parents' genetic code that the offspring can explore.

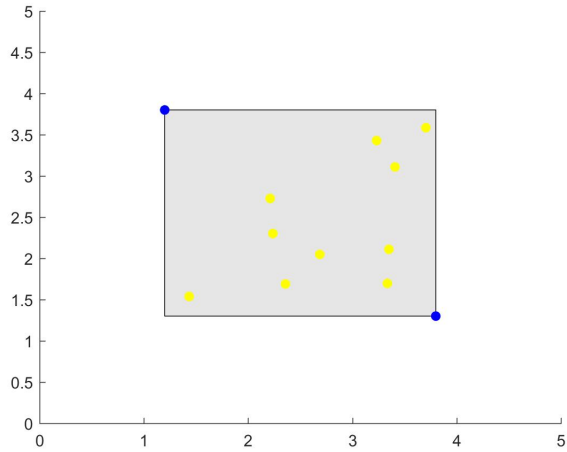


Figure 4: The blue points are the parents while the yellow are the offspring generated through the Arithmetical Crossover method (or Blend Crossover with $\alpha = 0$). The grey area represents the possible space where the offspring may be generated.

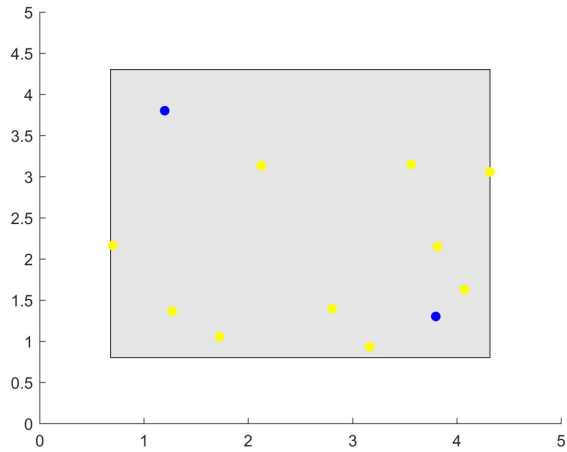


Figure 5: The blue points are the parents while the yellow are the offspring generated through the Blend Crossover method with $\alpha = 0.2$. The grey area represents the possible space where the offspring may be generated.

The last process is mutation. Mutation can be seen in nature as well, giving rise to features previously not present in the population. One way of mimicking this would be to for each individual in the population, with a probability of μ, replace them with a new, randomly generated individual.

By combining these processes the population overall tends to become a better solution to the minimisation problem for each generation. With elitism, only looking at the best performing individual, the cost function shall be non-increasing for each generation as the best individual or individuals are kept to for the next generation.

2.3.4 Particle Swarm

The Particle Swarm Optimisation (PSO henceforth) is another optimisation algorithm based on behaviour seen in nature. It is based on the swarm movement of animals such as flocks of bird or schools of fish. It initialises with a randomly generated swarm where each individual is given a random position and velocity. For each new iteration the particles' positions are updated according to their velocities:

$$x_i(t+1) = x_i(t) + v(t) \tag{12}$$

Each particle updates its velocity according to the acceleration it experiences. Each particle will accelerate towards its own historical best position and the best position of any particle so far [13]. In addition a random element is multiplied with the acceleration in each of the previously mentioned directions. The equation for updating the swarm's velocity and acceleration thus becomes:

$$\begin{aligned} v(t+1) &= \omega v(t) + a \\ a &= c_1 r_1 (p_g - x_i(t)) + c_2 r_2 (p_i - x_i(t)) \end{aligned} \tag{13}$$

where ω is the inertia weight, c_1 denotes the acceleration constant towards the global minimum and c_2 denotes the acceleration constant towards a particle's historical best position and r_1 and r_2 are uniform random numbers between zero to one. p_i denotes the i:th particle's best position. p_g is the best position of any particle.

Unlike the Nelder-Meads Simplex algorithm the PSO can utilise parallel computations. As in every generation all individuals are independent of each other each individual can be evaluated on a separate worker. The good thing is that as the number of worker grows the algorithm will get faster and by making sure the swarm size is a multiple of the number of cores the workers can be fully utilised at all times.

2.4 One-dimensional optimisation algorithms

For the position control gain parameters a one-dimensional search algorithm is required. As there are 66 parameters, there is no longer a real need to be able

to parallelise each optimisations itself. Instead all the different optimisation can be split up amongst the worker rather than splitting up the evaluations in each generation to make sure that each worker is utilised as much as possible.

2.4.1 Golden Section Search

The two ideas behind the method is that the search interval decreases at a constant rate α, more specifically $\alpha = \frac{1+\sqrt{5}}{2}$ to reuse one of the calculations from the previous iteration. The algorithm works as follows [4]: Given a search range from a_k to b_k, the function will be evaluated at λ_k and μ_k which are given by the following equation:

$$\begin{aligned} \lambda_k &= b_k - (b_k - a_k)/\alpha \\ \mu_k &= a_k + (b_k - a_k)/\alpha \end{aligned} \tag{14}$$

If $f(\lambda_k) < f(\mu_k)$ then the search interval is updated as follows

$$\begin{aligned} a_{k+1} &= a_k \\ b_{k+1} &= \mu_k \end{aligned} \tag{15}$$

and the new point at which the function will be evaluated will once again be given by Equation 14. If α is chosen to be $\alpha = \frac{1+\sqrt{5}}{2}$ the evaluation points will be given by:

$$\begin{aligned} \lambda_{k+1} &= b_{k+1} - (b_{k+1} - a_{k+1})/\alpha \\ \mu_{k+1} &= \lambda_k \end{aligned} \tag{16}$$

If however $f(\lambda_k) > f(\mu_k)$ the new search interval will be given by:

$$\begin{aligned} a_{k+1} &= \lambda_k \\ b_{k+1} &= b_k \end{aligned} \tag{17}$$

and the new points where the functions shall be evaluated will be given by:

$$\begin{aligned} \lambda_{k+1} &= \mu_k \\ \mu_{k+1} &= a_{k+1} + (b_{k+1} - a_{k+1})/\alpha \end{aligned} \tag{18}$$

As mentioned above, each iteration will decrease the search interval by α. If the desired size of the interval of uncertainty is L then the least required number of function evaluations n, are given by [4]:

$$\left(\frac{1}{\alpha}\right)^{n-1} \geq \frac{b_1 - a_1}{L} \tag{19}$$

2.4.2 Fibonacci Search

Fibonacci's search method is a search method similar to the Golden Section Search. However the search interval will not decrease at a constant rate, but rather decrease according to the Fibonacci sequence which is defined by:

$$\begin{aligned} F_k &= F_{k-1} + F_{k+2}, \quad k \leq 2 \\ F_0 &= F_1 = 1 \end{aligned} \tag{20}$$

The Fibonacci search method now instead modifies α to be given by [4]:

$$\alpha(n,k) = \frac{F_{n-k}}{F_{n-k+1}} \tag{21}$$

Now α is dependent of the two variables k and n. k is the same as in he golden section alorithm, the iteration number, and n is the total number of iterations the search will run for, which has to be specified beforehand. If the desired magnitude of the uncertainty is L then n will be required to fulfil:

$$F_n \geq \frac{b_1 - a_1}{L} \tag{22}$$

where a_1 and b_1 are lower and upper bound of the search interval respectively. From this equation the total number of iterations can be evaluated. Comparing this to the Golden Section Search, Fibonacci Search has a slightly better convergence rate for some intervals and desired search length, however Fibonacci search will require the number of iterations to be specified beforehand and calculating this from the final desired uncertainty will add some to the calculations required.

2.4.3 Binary Search

Commonly used as a search algorithm for sorted lists, although, in the section "Determining a cost function" it turns out that a version of binary search can be used to find the optimal values for the one-dimensional optimisation problem.

Binary search traverses sorted lists by looking at the value at the index in the middle of the search interval. From this value the algorithm determines whether to continue searching before or after the middle index thus splitting the search interval in half. From this new interval a new middle index is given and we can repeat the algorithm until the sought value is found. This means that the algorithm will split the interval in half for each comparison. The code for implementing this is shown below [1]:

```
BinarySearch(arr[1,2...N],val)

    low = 1
    high = N
    while (low <= high)
        mid = (high + low) / 2
        if (arr[mid] > val)
```

```
        high = mid - 1
    else if (arr[mid] < val)
        low = mid +1
    else
        return mid
```

However, the version needed for the optimisation is a last occurrence version binary search, which returns the largest index for which the sought value exists in. For example, on the following array, if the sought value is the last occurrence of 4, it would return the index 5.

$$\begin{bmatrix}1 & 1 & 4 & 4 & 4 & 6 & 6 & 8\end{bmatrix}$$

The algorithm described above only requires a few changes to instead look for the last occurrence. Now the algorithm cannot be aborted as soon as the correct value has been found, as it also needs to be verified that it is the last value. The new code can be seen below:

```
BinarySearch_Last(arr[1,2...N],val)

    low = 1
    high = N
    while (low <= high)
        mid = (high + low) / 2
        if (arr[mid] > val)
            high = mid - 1
        else
            low = mid + 1
    return low
```

As binary search reduces the search interval for each iteration and when looking for the last occurrence the algorithm will always have to divide the search interval until it is of size 1, therefore it will have to run for $\lfloor \log_2 n + 1 \rfloor$ iterations, where $\lfloor \rfloor$ is the floor function.

2.5 Flow dynamics of a check valve

A check valve is a directional valve, meaning that it allows the flow of the fluid in one direction while heavily restricting any flow in the opposite direction. In Figure 6 the check valve allows fluid to flow from A to B, although some leakage may occur that would allow small amount of fluid to flow from B to A. The flow rate will be assumed to be that of an orifice. The flow will also, for simplicity be assumed to be incompressible and laminar. That allows the volumetric flow rate to be obtained from Bernoulli's equation [9], which is given by the following equations:

$$
\begin{aligned}
q_v &= \frac{C_D A_{pass}}{\sqrt{1-\beta^4}} \sqrt{\frac{2}{\rho} \Delta p} \\
A_{pass}(p) &= \begin{cases} A_{leak} & \text{for } \Delta p \leq p_{crack} \\ A_{leak} + k(\Delta p - p_{crack}) & \text{for } p_{crack} < \Delta p < p_{max} \\ A_{max} & \text{for } \Delta p \geq p_{max} \end{cases} \\
\Delta p &= p_A - p_B \\
k &= \frac{A_{max} - A_{leak}}{p_{max} - p_{crack}}
\end{aligned}
\tag{23}
$$

where q_v is the volumetric flow rate through the check valve, p_A is the gauge pressure at A and p_B at B. p_{crack} is the pressure when the valve starts to open. p_{max} is the pressure required to fully open the valve. A_{pass} is the passage area that the fluid can flow in the check valve. A_{leak} is the cross sectional area that allows the fluid to leak through the valve in any direction when it is closed. A_{max} is the cross sectional area when the valve is fully open. C_D is the flow discharge coefficient, which is the ratio between the actual discharge and the theoretical discharge through a nozzle or orifice. ρ is the fluid's density and β is the ratio between the pipe diameter and the orifice diameter.

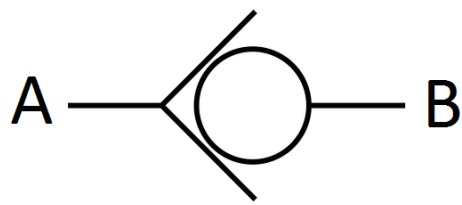


Figure 6: The schematic representation of a check valve

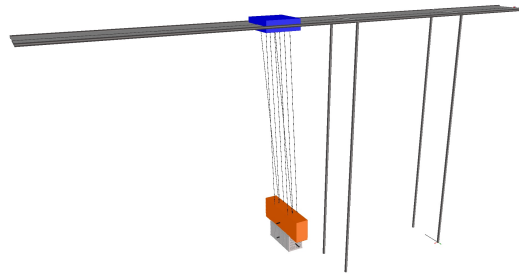


Figure 7: 3D visualisation of the model inside MapleSIM.

3 Method

To build the simulation environment MapleSIM and Simulink were used. MapleSIM is a Modelica-based physical modelling tool, with which the physical model of the STS crane was built. The physical model was then translated into C-code, with the help of MaplseSIM's code generations tools, and integrated with the controllers in Simulink. The then complete model in Simulink was used to run the algorithm in order to determine the parameters. As ABB already had a version of the physical model implemented in MapleSIM, but was lacking the dynamics of the cables and the hydraulics system, part of this project was to continue extending this model, to get a more accurate model of a crane.

3.1 MapleSIM Model

The MapleSIM model already provided had the cables the container is hanging from and the main frame of the STS-crane had been implemented along with the dynamics of these parts. A 3D model generated in MapleSIM can be seen in Figure 7. This model does not take into account the dynamics for the cables that are moving the trolley and the hydraulics where the cables are mounted, which was needed to be added.

3.1.1 Hydraulics mounting dynamics

The hydraulic cylinders are mounted as seen in Figure 8. The first idea was to build this using MapleSIM's standard hydraulics library and mounting using joint and rigid body frames from the multibody library. Unfortunately when merging this with the other model the simulations failed due to an internal error. This strategy was later abandoned for instead building a new model for the hydraulic cylinder and valves.

From the real schematic, a simplified version of the hydraulics was made, see Figure 9. In this simplified version the hydraulic cylinder is connected to two check valves, see Figure 9. These check valves together cause the hydraulic fluid to flow into the cylinder, extending it if the pressure drops below P_{low} and

Figure 8: Shows how the hydraulics is mounted on the STS-crane.

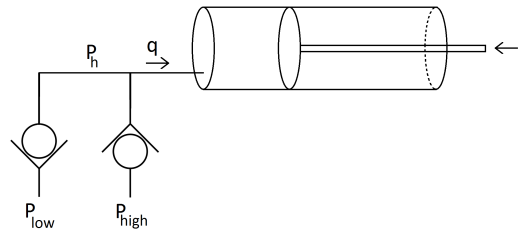


Figure 9: Simplified schematic over the hydraulic pipes and valve.

out of it, contracting it if the pressure exceeds P_{high}. Each of the valves are assumed have a crack pressure that is the same as the pressure required to fully open it, i.e. $p_{crack} = p_{max}$, since there is little knowledge of exactly how the check valves are constructed and is therefore assumed to be small in relation to the normal working pressure in the pipes. In Equation 23 the flow rate through a check valve is given. By combining two of these the flow to and from the hydraulic cylinder is given by:

$$q_v = \begin{cases} \frac{C_D A}{\sqrt{1-\beta^4}} \sqrt{\frac{2}{\rho}(P_{high} - P_h)} & \text{for } P_h > P_{high} \\ 0 & \text{for } P_{low} \leq P_h \leq P_{high} \\ -\frac{C_D A}{\sqrt{1-\beta^4}} \sqrt{\frac{2}{\rho}(P_h - P_{low})} & \text{for } P_h < P_{low} \end{cases} \tag{24}$$

A positive q_v means that the hydraulic fluid is flowing into the cylinder thus extending the hydraulics. The rate at which the cylinder extends depends on both the flow rate as well as the cross-sectional area of the cylinder and is given by:

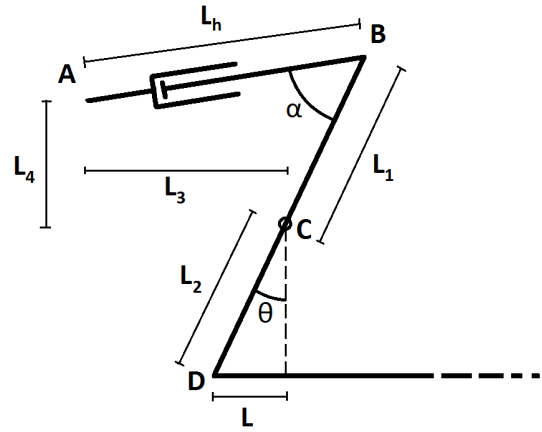


Figure 10: A simplified version of Figure 8 with the lengths defined. Point A and C are fixed revolute joints and B and D are free revolute joints.

$$\frac{dL_h}{dt} = \frac{q}{A} \tag{25}$$

$$P = \frac{F_h}{A} \tag{26}$$

The hydraulics are mounted as seen in Figure 8, a simplified schematic can be seen in Figure 10. If the lengths L_1, L_2, L_3 and L_4 are defined as seen in the figure along with the length of the cylinder, L_h, as well as the extension of the cable, L, the force on the hydraulics cylinder, F_h, can be calculated as follows:

$$F_h = F\frac{L_2}{L_1}\cos(\theta - \alpha)\cos(\theta) \tag{27}$$

$$\theta = \arcsin\left(\frac{L}{L_2}\right) \tag{28}$$

$$\alpha = \arctan\left(\frac{L_4 - L_1\cos(\theta)}{L_3 + L_1\sin(\theta)}\right) \tag{29}$$

while the extension of the cylinder, L_h causes the length L, to change according to Equation 30

$$\frac{dL}{dt} = \frac{L_h\sqrt{L_2^2 - L^2}}{L_1(L_4\sin(\theta) + L_3\cos(\theta))}\frac{dL_h}{dt} \tag{30}$$

$$L_h = \sqrt{(L_4 - L_1\cos(\theta))^2 + (L_3 + L_1\sin(\theta))^2} \tag{31}$$

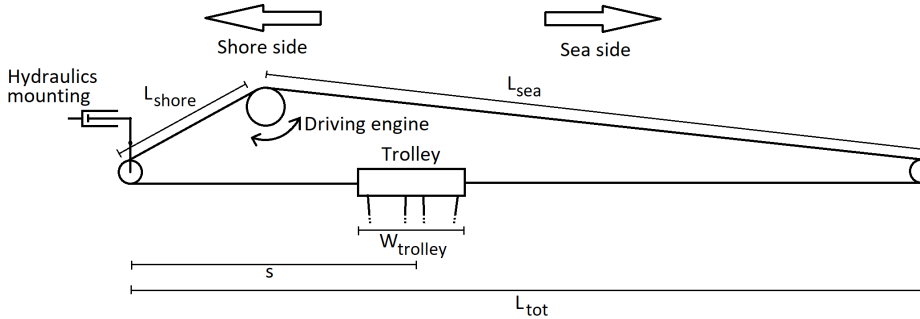


Figure 11: Simplified picture of the trolley drive system.

With equations 24 - 31 the dynamics of the point where the cables are attached, dL/dt, can be described by the force F, the cable extension L and cylinder length L_h.

3.1.2 Cable Model

The model also needed to take into account the elasticity in the cables that moves the trolley. As the stress is comparatively low the yield point of the cables, Young's modulus for the cables, will be assumed to be constant during the simulation. With this assumption cables will be approximated as spring-damper systems. Thus the force is given by Equation 32.

$$F = -k\Delta L - c\frac{d}{dt} \tag{32}$$

where k is the spring constant, c is the damping constant and ΔL is the extension of the cable defined as $\Delta L = L - L_0$, where L_0 is the relaxed cable length at a specific moment and L is the actual cable length at the same moment, see Figure 11. Since both L and L_0 are time dependant, the position of the trolley is controlled by varying L_0 is the way, $d\Delta L/dt$ is given by $d\Delta L/dt = \dot{L} - \dot{L_0}$. However since a cable only excerpts a force when it is stretched, the mass of the cable will be ignored as it is very small in comparison with the trolley. To accommodate this, the following modification was made:

$$F = -(k\Delta L + c\frac{d}{dt})\frac{1}{2}(\tanh(\beta\Delta L) + 1) \tag{33}$$

To create a differentiable step function, β is a value to increase the "sharpness" of the step and should therefore be rather large. Since the length of the cable varies over time so will k and c. Since k is given by:

$$k = \frac{F}{\Delta L} = \frac{A_c E}{L_0} \tag{34}$$

where A_c is the cross-sectional area of the cable, E is the Young's modulus of the cable and L_0 is the relaxed cable length. This causes k to vary as the

trolley moves. As the different cables are of different lengths the shore side cable will have a different k than the sea side cable. c is also dependent of the relaxed length of the cable. The equivalent dampening constant for dampers connected in series is:

$$\frac{1}{c_{eq}} = \sum_i \frac{1}{c_i} \tag{35}$$

Here c_i assumes the damping value for the cables per meter, which is constant over the whole cable. The equivalent dampening constant will therefore be approximated as:

$$c_{eq} = \frac{c}{L_0} \tag{36}$$

Similar to k, c will also change as L_0 is changing.

To control the position of the trolley in the simulation, the relaxed length of the cables will be controlled so as to simulate reeling in and out the cable on each side. On the shore side the relaxed total length is $L_0 = s + L_{shore} - W_{trolley}/2$, where s is the position of the trolley when the cables are relaxed and the other are simply defined in Figure 11. The cable on the sea side is implemented in a similar way. However, L_0 on the sea side is defined as $L_0 = L_{tot} - s - W_{trolley}/2 + L_{sea}$. Once again L_{tot} and L_{sea} are defined in Figure 11. If the relaxed length of each of the cables is defined as above the position of the trolley can be controlled by controlling s, physically, this is done by letting the driving engine wind the cable in either way.

For better control over the trolley, there is some tension in the cables at all time. The amount of this force to counteract the force from the minimum pressure in the hydraulic cylinder mentioned in previous section 3.1.1, since when the trolley has been still for some short time the valve separating the lower pressure reservoir and the cylinder opens and the hydraulic reverts to its vertical normal position. This tension has been mimicked by decreasing the relaxed length of the cables by ΔL_{shore} on the shore side and ΔL_{sea} on the sea side, thus shortening the cables on both sides. As the beam between B-D, as defined in Figure 10, is kept vertical when the crane is not used the amount that each side has to be shortened can be calculated from the following:

$$P_{min} A_c \frac{L_1}{L_2} = F_{shore} = k_{shore} \Delta L_{shore} = F_{sea} = k_{sea} \Delta L_{sea} \tag{37}$$

From this equation ΔL_{shore} and ΔL_{sea} are easy to evaluate.

3.2 Simulink model and Implementation into MATLAB

The Simulink model, which can be seen in the appendix B, has previously been made to emulate the control system on the STS cranes. By combining this with the MapleSIM model we have a complete system for simulating the crane with

controller. Not much has been changed with the Simulink model itself as it already is the same which is used for the real system. However, in order to be able to run the optimisation on this model some changes were required.

3.2.1 Integration with Optimisation algorithm

In order to be able to run the optimisation the Simulink model requires a few modification that allows for changing the control parameters from the workspace. By setting the parameters as global parameter they can be defined outside of Simulink and later used by the controller functions in the Simulink model.

As Simulink uses the variables in the current workspace, it will not be possible to use parallel computation as two different simulations will try to use the same values. However, this can be taken care of by using an instanced workspace that is specific for each simulation.

3.2.2 Cost function for X, Y, Z and Wcontrol parameter

When optimising over the control parameters X, Y, Z and $W$the main goal is to minimise the time it takes to come to a stop after the velocity reference becomes zero. Although to allow for intuitive control the position of where the trolley comes to a stop should be predictable, thus not sway back and forth and instead come to a stop in one smooth motion. Even if preventing this would result in a longer stabilising time it becomes more easier for the operator to steer the crane. Therefore we wish to have a cost function that penalises this behaviour in order to find control parameters that prevents the trolley from decelerating and then start accelerating again while stabilising, instead it should be one sustained deceleration. Examples of when the trolley decelerates and the accelerate before finally stabilising is shown in Figure 12.

As it is hard to determine what is the right combination of how fast a crane can stabilise and how it steers, a cost function would preferably be constructed from two parts; one part which increases as the stabilising time increases and one which increases the more the trolley switches between decelerates and accelerates. Furthermore, it should have a constant which allows to change the weight of each part. A proposed cost function is thus

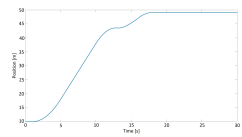


Figure 12: An example of an undesirable trolley trajectory, where the trolley comes to a stop and then starts moving again.

$$
\begin{aligned}
&cost(X, Y, Z, W) = p_1 + cp_2 \\
&p_1 = t_{stable} - t_{step} \\
&t_{stable} = min(t)\ such\ that\ |v(t)| < tol_v\ \forall t > t_{stable} \\
&p_2 = \int_{t \in a^-} a_{trolley}(t)\ dt
\end{aligned}
\tag{38}
$$

where $v(t)$ is the velocity of the container, t_{step} is the time at which the velocity reference goes to zero, tol_v is the accepted uncertainty in velocity, $a(t)$ is the acceleration of the trolley and a^- is the time when the acceleration of the trolley is negative except for the last sustained declaration that stabilises the container.

3.2.3 Cost function for the position control gain parameter

The position control gain parameter, $P_c^{I_{rl}, I_d}$, are in use when there is no interactions with a human driver, as the controller using the position control gain parameters is only used when while the crane is driving automatically. This means that the time it takes to stabilise at a certain location is the only parameter that is needed to be taken into consideration, therefore, another cost function is needed to optimise these control parameters.

A simple cost function can be constructed similar to the first part in the previous cost function, Equation 38.

$$
\begin{aligned}
&cost(P) = t_{stable} - t_{step} \\
&t_{stable} = min(t)\ such\ that\ |y(t) - r(t)| < tol_p\ \forall t > t_{stable}
\end{aligned}
\tag{39}
$$

where P is a vector of control parameter for the simulation, $y(t)$ is the position of the container in meters, $r(t)$ is the desired position for the container and t_{step} is the time at which the change in reference occurs. Albeit a very simple cost function, since it contains the only property that is desired to be minimised, it may have multiple local minima. Therefore a few cost functions will be proposed and tested to find one which has a minimum at the same location as the cost function (39) and which is convex. Since there is no need to take the human-machine interface into consideration this section will only focus on getting a cost function that is as close to convex as possible. Even though the human-machine interface does not prevent convexity it does add another element to the penalty function, making it harder to achieve convexity.

The proposed cost functions are:

$$
\begin{aligned}
&cost(P) = \int_{t_{step}}^{t_{sim}} (y(t) - r(t))s(t)dt \\
&s(t) = \begin{cases} 1 & \text{for } |y(t) - r(t)| > 0.10 \\ 0 & \text{for } |y(t) - r(t)| \leq 0.10 \end{cases}
\end{aligned}
\tag{40}
$$

where t_{sim} is the time at which the simulation ends, the rest is same as defined above.

$$\begin{aligned} cost(P) &= \int_{t_{step}}^{t_{sim}} (y(t) - r(t))s(t)n^{(P(t))}dt \\ s(t) &= \begin{cases} 1 & \text{for } |y(t) - r(t)| > 0.10 \\ 0 & \text{for } |y(t) - r(t)| \leq 0.10 \end{cases} \end{aligned} \tag{41}$$

$P(t)$ is the number of times the output has passed the reference point until the time t and n is a number greater than 1 determining how much to punish each pass. This is to prevent the load from swinging past the reference point.

Here the cost function (39), may have multiple local minima as can be seen in Figure 13. Cost function (41) yields a cost function that will be easier to optimise on, see Figure 14. The global minimum for each of the cost functions are located at the same location for almost all different cost functions of and differs at most 0.01 from the optimal value of (39). It eliminates most local minima and allows both Fibonacci and golden section search to work well. The plots seen in figures 13 - 14 are just a handful from each cost function.

However, from analysing the plots in Figure 13 there is another way to optimise the parameter, as it can be seen that the cost function decreases until it sharply rises. This is then repeated as can be seen in most of the plots in Figure 13, and in particular we can see it in the plot where $I_{rl} = 4$ and $I_d = 3$. This is due to the container overshoots the reference and just passes outside the tolerated error, $r \pm tol_p$, see Figure 15. Since the cost function is the time for which all later y are within $r \pm tol_p$, the one that overshoots get a significantly higher cost.

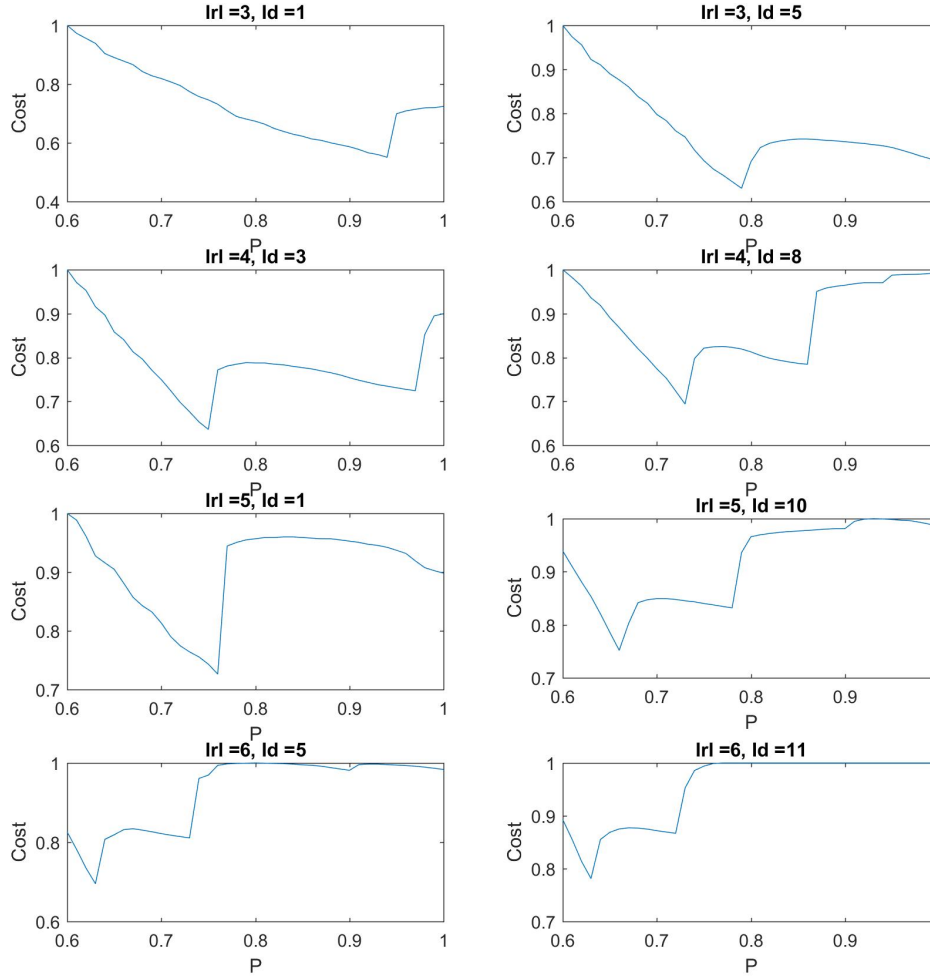


Figure 13: The cost from the cost function in (39) plotted against the value of $P_c^{I_{rl},I_d}$. All costs has been normalised.

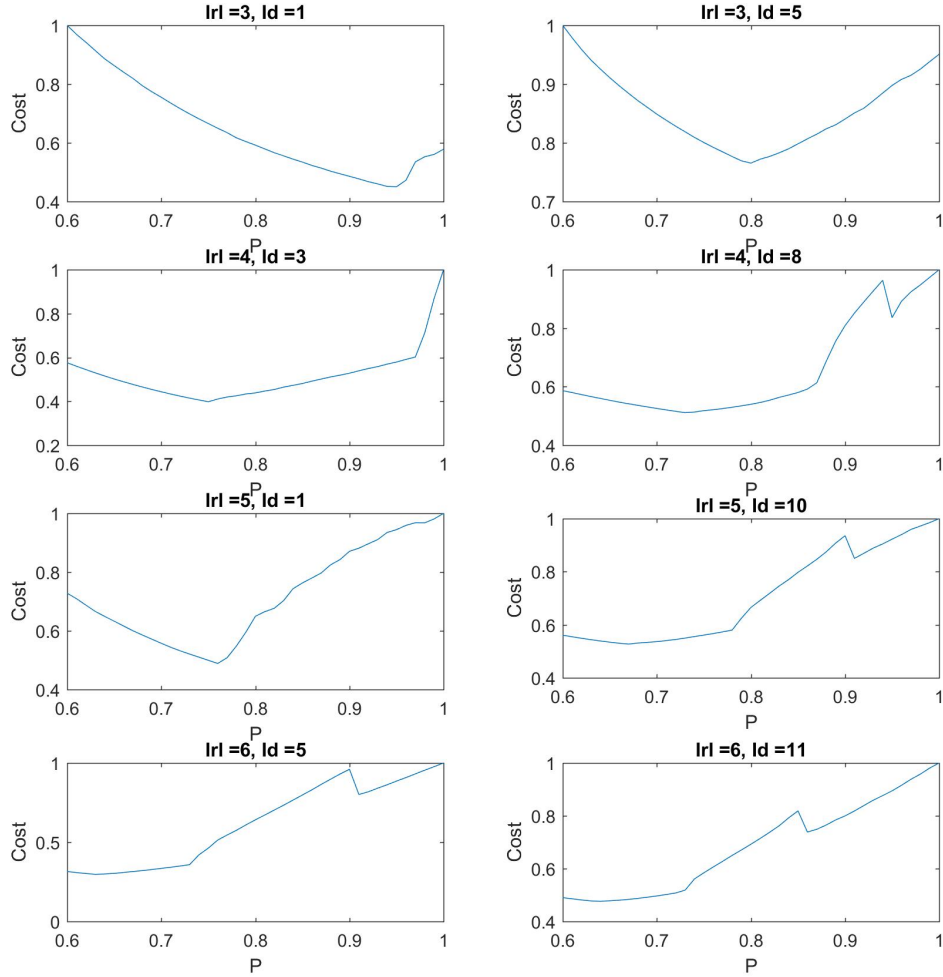


Figure 14: The cost from the cost function in (41) plotted against the value of $P_c^{I_{rl},I_d}$. All costs has been normalised.

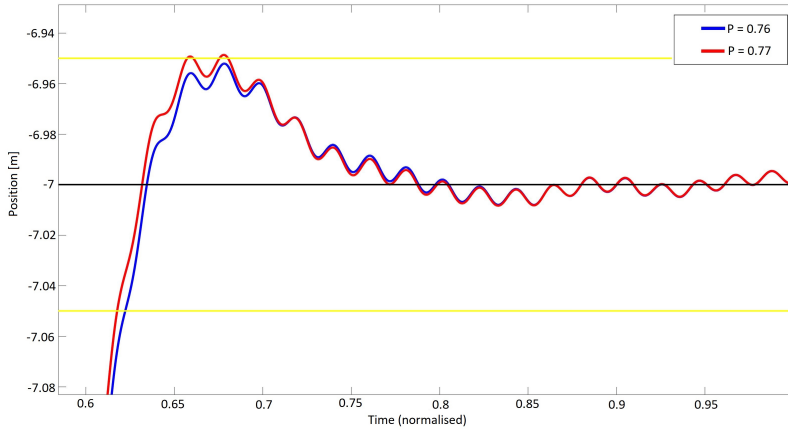


Figure 15: Two trajectory where the control parameter are similar, 16 (blue) and 17 (red) , but they yield very different costs since the red one overshoots by a small margin.

If the number of times the trajectory passes the limits $r \pm tol_p$ are plotted against the value of the control parameter, see Figure 16, we notice that it is non decreasing, meaning a higher value leads to a more aggressive controller and the risk of overshooting the target location increases. Therefore another way to minimise (39) would be to find the highest value for the control parameter that does not overshoot the target. We can thus construct the cost function:

$$cost(P) = n \tag{42}$$

where n is the number of times $y(t)$ has left the region $r \pm tol_p$.

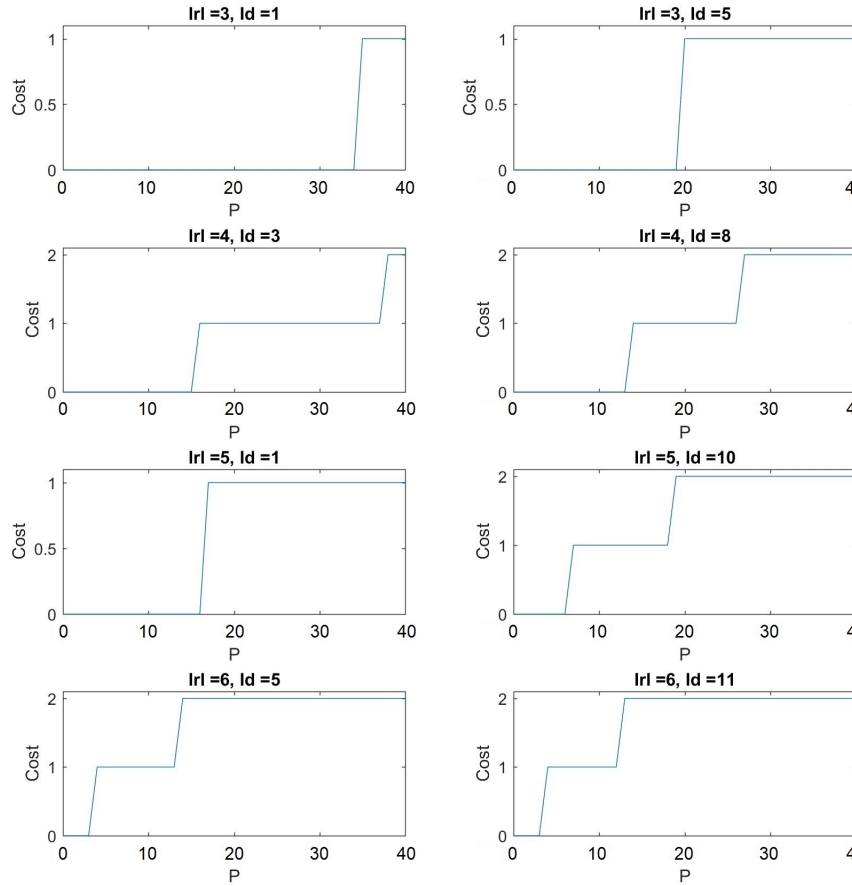


Figure 16: Number of times y passes the $r \pm tol_p$

3.2.4 Choice of 4D optimisation algorithm

To decide which optimisation function to use the different methods were tested on a few different test functions. Since not much is know about the properties of the cost function the optimisation algorithm has to be tuned and improved on multiple test functions to make it as robust as possible. Commonly used test functions will be used to test the convergence rate of the optimisation algorithm and how it performs on functions with multiple local minima. The functions, taken from Simon Fraser Virtual Library of Simulation Experiments [12], that will be used for testing are:

- Ackley Function

$$f(x) = -a \exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}\right) - \exp\left(\sum_{i=1}^{d} \cos(cx_i)\right) \tag{43}$$

$$\text{a} = 20,\ \text{b} = 0.2,\ \text{c} = 2\pi$$

- Levy function N.13

$$f(x) = 10d + \sum_{i=1}^{d} x_i^2 - 10\cos(2\pi x_i) \tag{44}$$

- Styblinski–Tang function

$$f(x) = \frac{1}{2}\sum_{i=1}^{d} x_i^2 - 16x_i^2 + 5x_i \tag{45}$$

- Michalewicz function

$$f(x) = -\sum_{i=1}^{d} \sin(x_i)\sin\left(\frac{ix_i^2}{\pi}\right)^{2m} \tag{46}$$

$$\text{m} = 10$$

- Hartmann function

$$f(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{d} A_{ij}(x_j - P_{ij})^2\right) \tag{47}$$

$$\alpha = \begin{pmatrix}1.0 & 1.2 & 3.0 & 3.2\end{pmatrix}^T$$

$$A = \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}$$

$$P = 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix}$$

Each of the optimisation algorithms were tested 1000 times on each function were the initial swarm, population or simplex is randomly chosen each time. In order to be able to more easily compare the different algorithms, a new measure has been added in the last row of each table. This measure is a "crude" estimate of the number of function evaluations required to reach a success rate of 99%, which has been calculated with the following equation:

$$N_{0.99} = N \frac{\ln(0.01)}{\ln(1-r)} \tag{48}$$

where N is the average number of evaluations and r is the success rate.

The performance of the optimisation algorithms on the test functions are the following:

Ackley function

	PSO	GA	Simplex
Success Rate	99.1 %	96.7 %	5.8 %
Average Eval	733.0	998.7	59.9
Nbr Eval for 99 %	716.6	1348.2	4620.0

Styblinski–Tang function

	PSO	GA	Simplex
Success Rate	80.5 %	84.5 %	15.2 %
Average Eval	636.6	932.5	54.5
Nbr Eval for 99 %	1793.2	2303.3	1521.6

Levy function N. 13

	PSO	GA	Simplex
Success Rate	97.6 %	93.7 %	47.4 %
Average Eval	515.7	644.9	42.1
Nbr Eval for 99 %	636.8	1074.3	301.6

Hartmann function

	PSO	GA	Simplex
Success Rate	73.3 %	77.2 %	14.7 %
Average Eval	736.6	1169.2	73.9
Nbr Eval for 99 %	2568.7	3642.0	2141.8

Michalewicz's function

	PSO	GA	Simplex
Success Rate	99.5 %	99.3 %	19.4 %
Average Eval	744.1	1064.5	66.2
Nbr Eval for 99 %	646.7	988.0	1414.5

For the multidimensional optimisation the PSO was chosen. It performed better than the GA and was more consistent on the test functions than Nelder-Mead Simplex and also because it is better at utilising multiple cores for parallel computing.

3.2.5 Improving the 4D optimisation

The success rate of the optimisation function is currently low on the Hartmann function and the Styblinski-Tang function. To increase the success rate a larger swarm size is needed, although this will increase the number of function evaluations, which is, as previously mentioned, more important to keep at a minimum. One way to reduce the number of function evaluations would be to remove particles that are close to each other as there is no real need to evaluate two similar positions. Thus a particle can be removed if it is within the accepted error of another particle i.e.

$$\text{Remove particle if } \sum_i \left(\frac{x_i}{tol_i}\right)^2 < 1 \tag{49}$$

where x is the position of the particle and tol_i is the error tolerance along the i-axis. This can sometimes be a problem as this will remove a particle that is close to another but has different velocity and that would explore a different area. Thus it is likely to lower the success rate as well. This will be tested on the Hartmann function as there is more room for improvement than there is on the other test functions. The error tolerance has been set to mimic the same error tolerance as in the 4D optimisation problem. The results can be seen in table 3.2.5 below:

Hartmann 4D function

PSO	Opt.1	Opt.2
Success Rate	76.9 %	77.7 %
Average Eval	675.9	874.6
Nbr Eval for 99 %	2124.1	2684.0

Table 1: Option 1 removes a particle if it is too close to another particle. Option 2 leaves them as they were. Both have a swarm size of 80

As expected the number of evaluations is reduced, even if there is a slight reduction in the success rate. Using the same measure as before, as defined in Equation 48, removing particles that are too close to other particles proves worth it.

Since we do not require a resolution better than the error tolerance of each variable we can discretize each axis. This will make the search space finite which may make it easier to find the minimum and possibly also lower the number of function evaluations required.

Hartmann 4D function

PSO	Opt.1	Opt.2
Success Rate	76.7 %	76.4 %
Average Eval	630.5	676.3
Nbr Eval for 99 %	1993.3	2156.8

Table 2: Options 1 uses discrete states. Options 2 uses continuous states. Both remove particles that are too close to other particles

The success rate did not increase and the number of evaluations did not decrease significantly. However, with the discrete state some of the particles will end up in previous evaluated states. By storing all function evaluation in a lookup table and using that instead of evaluating the number of function evaluations can be decreased further, this time without compromising the success rate.

Hartmann 4D function

PSO	Opt.1	Opt.2
Success Rate	78.4 %	78.9 %
Average Eval	475.6	633.3
Nbr Eval for 99 %	1429.2	1874.4

Table 3: Opt. 1 uses discrete states and saves all evaluated states to recall if it needs to evaluate it again. Opt. 2 evaluates the function every time regardless.

Reusing old function evaluations with the discrete states does reduce the number of function evaluations by a fourth but leaves the success rate unaffected. With these changes to the PSO, the number of function evaluations can be reduced greatly.

3.2.6 Choice of 1D optimisation algorithm

Due to the discovery in section 3.2.3, it is enough to find the highest value of the control parameter that does not overshoot $r \pm tol_p$. Since the cost function Equation 42 is non decreasing it is possible to use a binary search method that finds the last occurrence, which has a slight advantage to both the Fibonacci and the golden sections search algorithms. Assuming we require an error tolerance of 1 the number of function evaluations from using a binary search method would be:

$$\left\lfloor log_2\left(\frac{40-0}{1}\right)+1\right\rfloor = \lfloor log_2(41)\rfloor = \lfloor 5.37\rfloor = 5 \tag{50}$$

Using the same interval and error tolerance the number of evaluations for the golden section search method 19 is given from the following equations:

$$\begin{aligned} \left(\frac{1}{\alpha}\right)^{n-1} &\geq \frac{40-0}{1} = 40 \\ n &> 1 + \frac{ln(40)}{ln(\frac{1}{\alpha})} = 8.66... \longrightarrow n = 9 \end{aligned} \tag{51}$$

and for the Fibonacci search the number of function evaluations 22 is given by the following equation:

$$F_n \geq \frac{40-0}{1} = 40 \longrightarrow F_n = 55 \longrightarrow n = 9 \tag{52}$$

where F_n is the Fibonacci sequence defined in Equation 20. Meaning that in this case both the golden section search and Fibonacci algorithms will need the same number of function evaluations, which compared to the binary search method is almost twice as many. For this case the binary search method outperform the other algorithms and will therefore be used for the one-dimensional optimisation.

3.2.7 Problems exporting into MATLAB

When the MapleSIM model was exported to MATLAB the model became unstable. With the help from MapleSIM support a solution to this problem was found. The issue was partly attributed to a bug in MapleSIM, not allowing π to be set as a symbol in MapleSIM, and instead using its numerical value. Another problem was that one of the gain-blocks was set to 0, which resulted in that an equation being removed during the simulation. However, correcting these problems did not solve the issue entirely. Although after copying the changes made to the model into an older model the problems seem to be solved. The exact cause of why this fixed the issues is not known. In later versions of MapleSIM new functionality allow the user to easier analyse the problem more closely.

Another problem that has arisen in the model when combining it with the controllers is illustrated in Figure 17. This only happen for $I_{rl} = 1$ and $I_{rl} = 2$. The trolley slams into one of the ends of the track and afterwards does not register the error and does not correct it. Therefore it will not be possible to determine the parameters for $I_{rl} = 1$ and $I_{rl} = 2$.

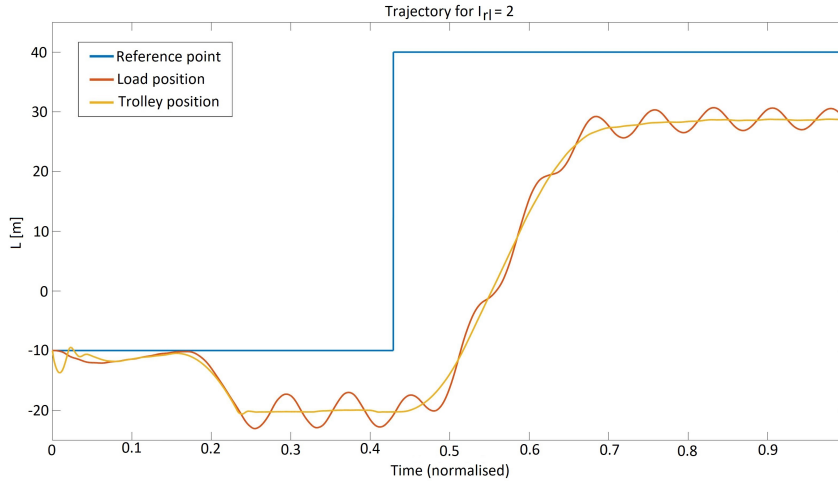


Figure 17: Shows the trajectories for the load and trolley from the simulation where $I_{rl} = 2$.

4 Results

4.1 Model output and its validity

In Figure 18 is the comparison between a simulation run in MapleSIM and one run in Simulink and the difference between the two. These are done without any controller and with the same input to show the deviation between the models.

In Figure 19 is the trajectory for the trolley and the load generated in Simulink along with the regulator.

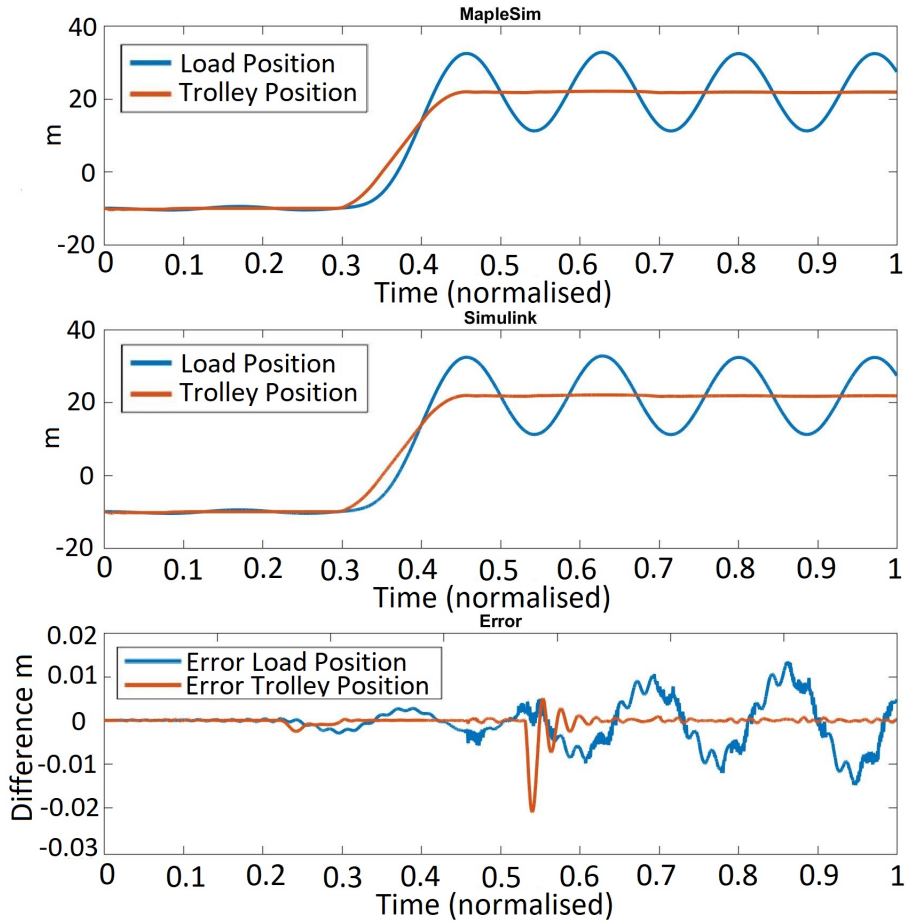


Figure 18: The difference between the same model simulated in MapleSIM compared to Simulink.

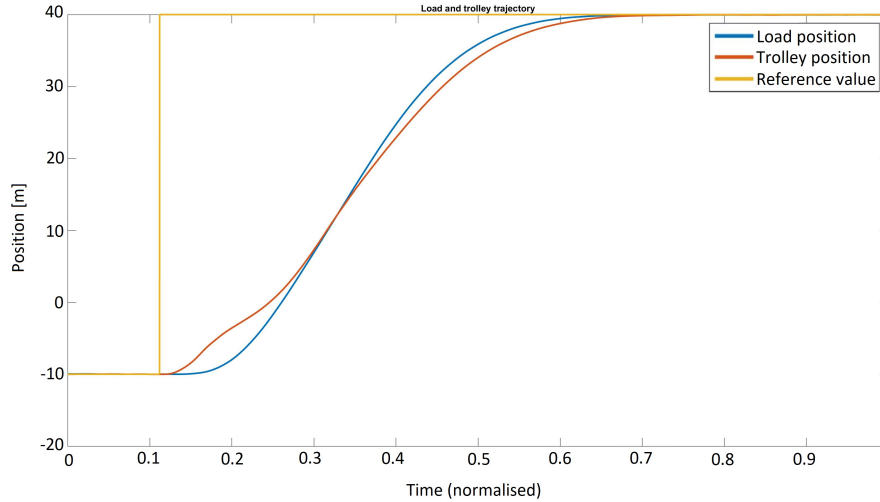


Figure 19: The trajectory for the load (blue) and trolley (red).

In Figures 20 and 21 we see the elongation in the cables driving the trolley, ΔL in the Equation 37.

Figure 22 shows the movement of the hydraulic. In figures 20 – 22 the time step takes place at $t = 0.1$. All of them are taken from MapleSIM simulation. Therefore there is no controller dampening the sway of the load.

Figure 23 shows the trajectories differences for the position of the trolley (red) and the load (blue) respectively, for the simulation with and without the hydraulics dynamics activated.

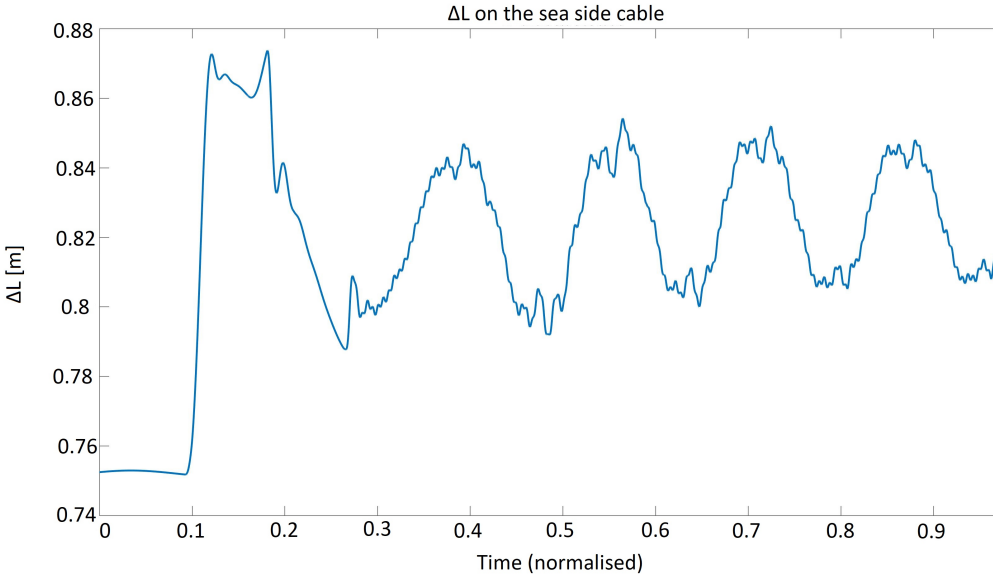


Figure 20: Stretching on the sea side cable.

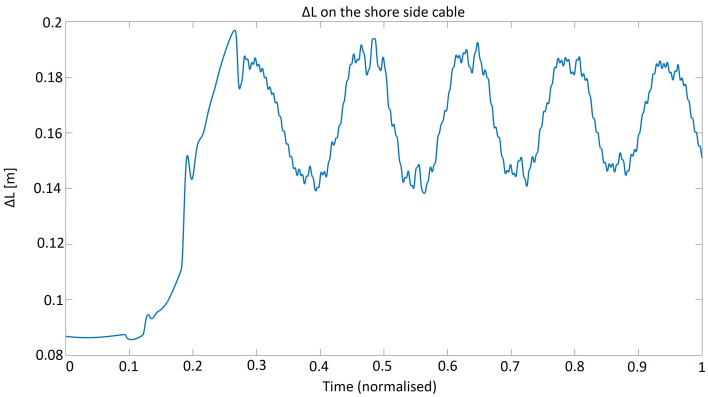


Figure 21: Stretching on the shore side cable.

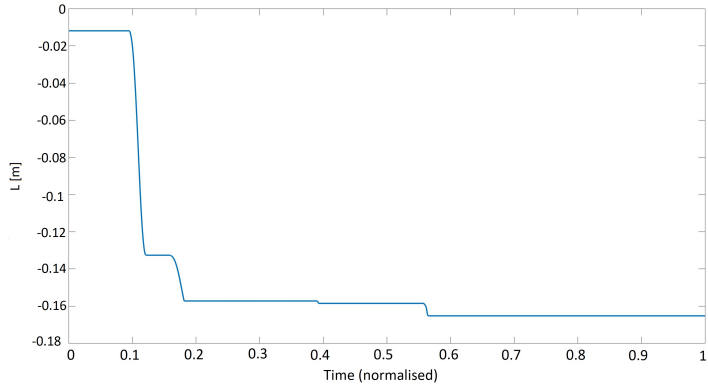


Figure 22: Shows the movement of the hydraulic mounting.

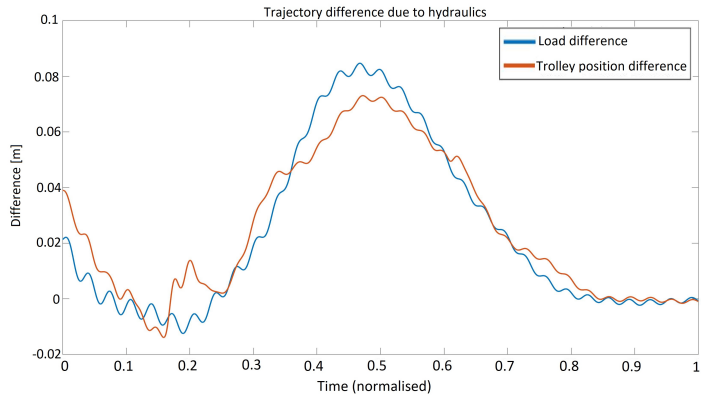


Figure 23: The difference in trajectory due to the effects of the hydraulics. The red is the difference in position for the trolley and the blue is the difference in position for the load.

4.2 Optimisation algorithms performance

The time it takes to run the optimisation for all parameters is around 14 hours, running in total around 13000 simulations, on a laptop with the following hardware.

- CPU: Intel Core i7 - 4710HQ (4 cores @ 2.50 GHz)
- RAM memory: 16 GB
- Operating system: 64-bit Windows 10 home

This is acceptable run time since if the optimisation is started at the end of a workday it should have finished by the next day.

4.3 Comparison to manually determined parameters

The control parameters given by the optimisation on the model are:

$$\mathrm{X} = \begin{pmatrix} - & - & 2.0 & 1.8 & 1.6 & 2.0 \end{pmatrix}$$

$$\mathrm{Y} = \begin{pmatrix} - & - & 2 & 2 & 2 & 2 \end{pmatrix}$$

$$\mathrm{Z} = \begin{pmatrix} - & - & 3.0 & 2.0 & 1.0 & 1.0 \end{pmatrix}$$

$$\mathrm{W} = \begin{pmatrix} - & - & 3.5 & 2.8 & 4.2 & 1.4 \end{pmatrix}$$

$$\mathrm{P}_c = \begin{pmatrix} - & - & - & - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - & - & - & - \\ 20 & 19 & 15 & 15 & 15 & 15 & 13 & 12 & 14 & 17 & 14 \\ 15 & 11 & 11 & 11 & 11 & 11 & 11 & 11 & 10 & 11 & 11 \\ 9 & 8 & 6 & 5 & 5 & 6 & 6 & 6 & 6 & 6 & 6 \\ 6 & 4 & 2 & 2 & 2 & 2 & 2 & 2 & 3 & 3 & 2 \end{pmatrix}$$

compared to the parameters used for a real crane which are:

$$\mathrm{X} = \begin{pmatrix} 1.65 & 1.6 & 1.64 & 1.64 & 1.68 & 1.68 \end{pmatrix}$$

$$\mathrm{Y} = \begin{pmatrix} 3 & 3.5 & 5 & 5.5 & 5.5 & 5.5 \end{pmatrix}$$

$$\mathrm{Z} = \begin{pmatrix} 9.0 & 9.6 & 10.0 & 10.0 & 14.0 & 14.0 \end{pmatrix}$$

$$\mathrm{W} = \begin{pmatrix} 2.1 & 2.1 & 2.1 & 2.1 & 2.1 & 2.1 \end{pmatrix}$$

$$\mathrm{P}_c = \begin{pmatrix} 24 & 22 & 18 & 16 & 16 & 16 & 16 & 16 & 16 & 16 & 16 \\ 24 & 22 & 18 & 16 & 15 & 15 & 15 & 16 & 16 & 16 & 16 \\ 20 & 20 & 18 & 16 & 15 & 15 & 15 & 16 & 16 & 16 & 16 \\ 20 & 16 & 14 & 14 & 14 & 14 & 14 & 14 & 13 & 13 & 13 \\ 13 & 13 & 10 & 10 & 10 & 10 & 10 & 8 & 6 & 6 & 6 \\ 15 & 13 & 10 & 4 & 4 & 4 & 4 & 4 & 0 & -4 & -4 \end{pmatrix}$$

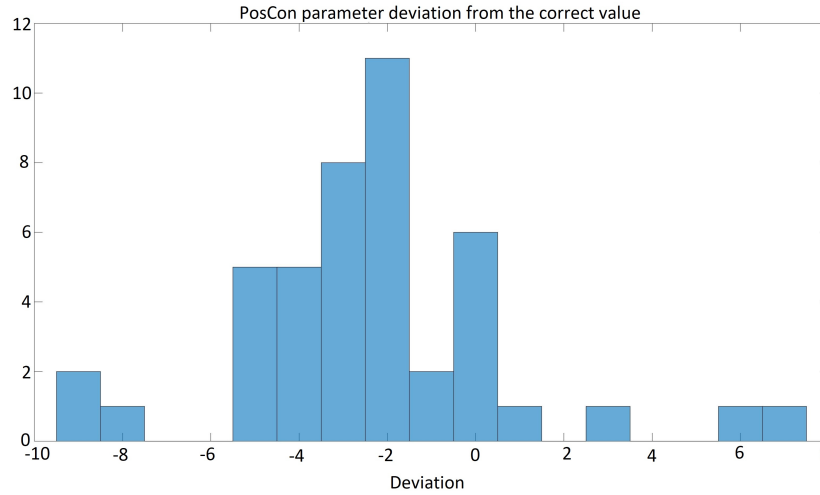


Figure 24: Histogram showing the distribution of differences between the solution to the optimisation problem and the real value for the `PosCon` parameter.

If the first two rows in the value matrices for the parameter P_c are excluded, as the simulation failed to optimise these, a histogram over the difference between the solution to the optimisation problem and the values of the real crane can be seen in Figure 24. The average value for the error is -2.32, i.e. the value from the optimisation algorithm is lower than the values currently used.

5 Discussion and Conclusion

One of the big problems in this thesis is that there has not been any real verification of the MapleSIM model against data from real cranes. The only verification is to compare the output to what the supervisor, Jonas Öhr, has observed. However, the model still catches some of the effects of a real crane compared to an simpler model of a pendulum on a stiff rod.

For the 4-dimensional optimisation algorithm, the number of function evaluations was reduced by half on Hartmann function, although the improvements did not increase the success rate. For the one-dimensional optimisation a cost function that allowed for a deterministic search method was found. The last occurrence binary search method proved to decrease the number of function evaluations required to find the optimal the position control gain parameters by 30%. Furthermore, as the method is deterministic it will always find the correct minimum if the observations are true for all possible cases, which seems to be a fair assumption under the current conditions.

When comparing the parameter in section 4.3, it is clear that there are a lot of differences between the manually tuned values for X_i, Y_i, Z_i and $W_i$and the ones given by the model. However, the values for the position control gain parameters shows closer resemblance. With an average deviation of -2.32 these parameters will probably see use. The fact that they are lower means that the optimisation algorithm is more cautious, as lower value for the control parameters are less likely to swing past the reference. So if the error tolerance in the position of the container had been increased the mean error would be reduced.

Apart from determining the position control gain parameters it will also be a good tool to analyse how the parameters change when parameters of the model change, and even though they may not be accurate it can at least provide indication of how, i.e. the hydraulics, affects the control parameter.

5.1 Future work

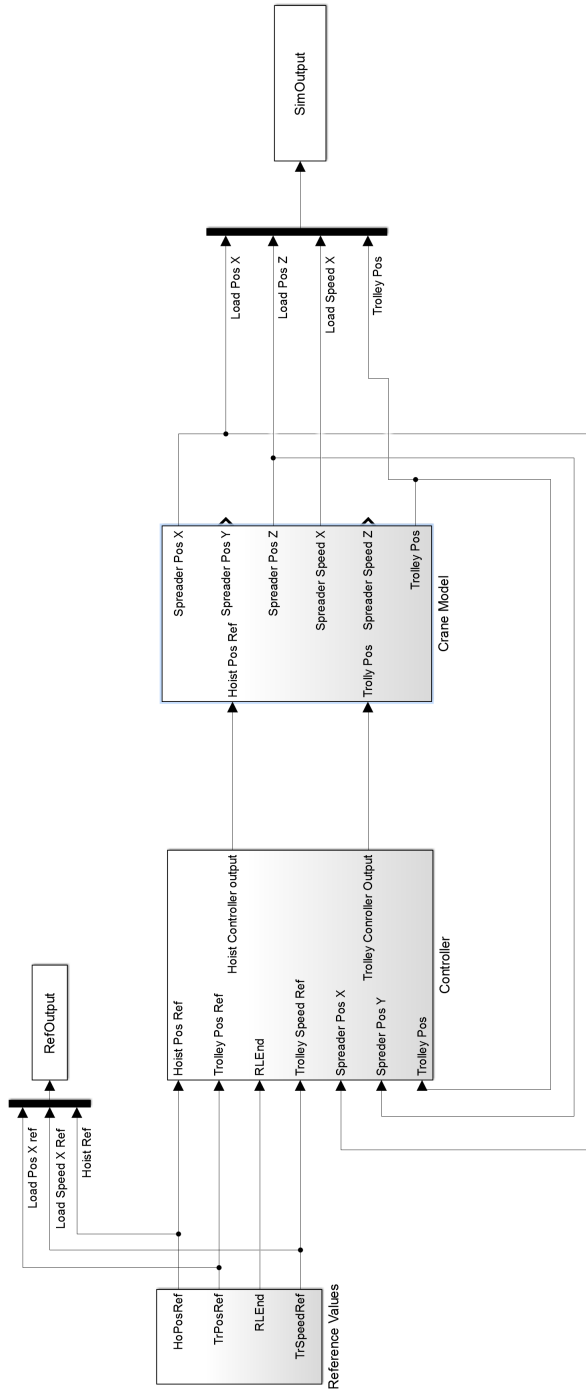
In order to improve the final parameters, improvements to the MapleSIM model would be the first place to start. To improve the crane model, one should collect data from different cranes around the world and use that to improve the model. Since the optimisation algorithm performs well enough, it can find the optimal parameters for the given model and search interval within a reasonable time frame.

Another improvement would be to find what is causing the results shown in Figure 17, as this prevents the script from being able to determine the control parameters for $I_{rl} = 1$ and $I_{rl} = 2$.

A Flow Chart for the Simplex Method

Figure 25: Flow chart for Nelder-Mead simple algorithm. This flowchart has been adapted from [10]

B Simulink model



References

[1] C. Pal A. Oommen. "Binary Search Algorithm". In: *International Journal of Innovative Research in Technology* 1.5 (Oct. 2014), pp. 800–803.

[2] Crane systems ABB. URL: http://new.abb.com/ca/about/technology/crane-systems (visited on 04/02/2017).

[3] U. Bodenhofer. *Lecture Notes; Genetic Algorithms: Theory and Applications*. Dept. of Knowledge-based Mathematical Systems, Johannes Kepler University, Linz, Austria, 2003/2004.

[4] L.-C. Böiers. *Mathematical Methods of Optimization*. Lund, Sweden.: Studentlitteratur AB, 2010. ISBN: 9789144070759.

[5] D. J. Leith and W. E. Leithead. "Survey of gain-scheduling analysis and design". In: *International Journal of Control* 73.11 (Jan. 2000), pp. 1001–1025. DOI: 10.1080/002071700411304. URL: https://doi.org/10.1080/002071700411304.

[6] A. John Arul S. Rajeswari K. K. Kuriakosa S.A.V. Satya Murty M. Mehra M.L. Jayalal. "Study on Different Crossover Mechanisms of Genetic Algorithm for Test Interval Optimization for Nuclear Power Plants". In: *International Journal of Intelligent Systems and Applications* 6 (Jan. 2014), pp. 20–28. DOI: 10.5815/ijisa.2014.01.03.

[7] MapleSim. URL: https://www.maplesoft.com/products/maplesim/ (visited on 04/02/2017).

[8] Inc MathWorks. URL: http://www.mathworks.com (visited on 04/02/2017).

[9] R.W. Miller. *Flow measurement engineering handbook*. Chemical engineering books. McGraw-Hill, New York, NY, United States, 1996. ISBN: 9780070423664. URL: https://books.google.se/books?id=0e9RAAAAMAAJ.

[10] J. A. Nelder and R. Mead. "A Simplex Method for Function Minimization". In: *Computer Journal* 7 (1965), pp. 308–313. URL: http://www.bibsonomy.org/bibtex/2053fb791805bd1debd80a198e8f3e45c/brian.mingus.

[11] A. Popov. "Genetic algorithms for optimization". In: *Programs for MATLAB.* Hamburg, Germany. (2005).

[12] S. Surjanovic and D. Bingham. *Virtual Library of Simulation Experiments, Simon Fraser University*. Burnaby, Canada. Dept. of Statistics and Actuarial Science, Accessed: 2016-05-20. URL: https://www.sfu.ca/~ssurjano/optimization.html.

[13] Y. Zhu W. Wang Y. Zhou X. Li. "A discrete particle swarm optimization algorithm applied in constrained static weapon-target assignment problem". In: *2016 12th World Congress on Intelligent Control and Automation (WCICA)* (June 2016). Guilin, China, pp. 3118–3123. DOI: 10.1109/WCICA.2016.7578704.

Lund University
Department of Automatic Control
Box 118
SE-221 00 Lund Sweden

Document name
MASTER'S THESIS

Date of issue
December 2017

Document Number
TFRT- 6071

Author(s)
Hampus Hellström

Supervisor
Jonas Öhr, ABB Crane system
Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden
Rolf Johansson, Dept. of Automatic Control, Lund University, Sweden (examiner)

Title and subtitle
Automatic Procedure for Determining Control Parameters for Ship-to-Shore Cranes

Abstract

The objective of this thesis has been to create a tuning functionality that can determine the control parameters for a crane that loads and unloads containers from ships. To accomplish this a model of the crane has been developed in MapleSIM and exported into MATLAB's Simulink. In MATLAB cost functions have been developed and evaluated to later be used in optimisation algorithms to find the optimal parameters.

There are two sets of control parameters that the script needs to determine, the first is composed of 6 subsets of 4 co-dependent parameters that need to be optimised together. The other set consists of 66 parameters that are independent of each other. A few different optimisation methods have been considered, but for the first set a particle swarm optimisation was used and for the second set, due to the cost function, a form of binary search was possible to use.

As concluding results, we found that the underlying model of the crane was not accurate enough to determine the first set of control parameters. The second set of control parameters are similar enough to the real values and can possibly be used on a real crane.

Keywords

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title
0280-5316

ISBN

Language
English

Number of pages
1-45

Recipient's notes

Security classification

http://www.control.lth.se/publications/