SUMMARY

A thrust allocation system is used to determine how the desired forces, computed by a high level control sytem, can be distributed among the thrusters. The main goal of the thrust allocation is to obtain the desired force, but other objectives can also be included. Such secondary goals can be to minimize fuel consumption, keep wear and tear of the thruster to a minimum and avoid overloading the power systems. The thrust allocation should also take forbidden sectors and actuator rate constraints into account. It is essential to safe operation that the allocation system provides a solution, and provides the solution in time.

In this thesis MPC (Model predictive control) is suggested as a method to solve the control allocation problem for CyberRig I (a scaled model of a semi-submersible drilling unit). 3 MPC algorithms are simulated in matlab, and the most complete are chosen for on-line implementation. The algorithm is based on an extended thrust formulation, and allows for rotatable thrusters. The cost function penalizes change in thust magnitude and in the azimuth angle. Forbidden sector constraints and rate constrains, both for thrust magnitude and angle, are implemented.

It is shown in simulations that the MPC algorithm performs well in comparison with an existing quasi-static method. Its main benefit over the quasi-static method is the ability to handle constraints. The cost of using MPC is increased computational efforts.

PREFACE

This work is the concluding part of the Master Degree at Norwegian University of Technology and Science (NTNU), Department of Engineering Cybernetics.

I would like to thank Jørgen Spjøtvold for coming up with the idea to the thesis and serving as my supervisor throughout this semester. His views, ideas and help have been of great value to the development of my assignment. I also want to thank my supervisor Tor Arne Johansen for his guidance through this project, especially in the initial phase and the finishing phase. Stefani Bertelli has helped me with the practical issues with CyberRig I, for this I am thankful. Lastly I would like to thank my colleagues and class mates for creating a constructive and fun working environment.

Trondheim, Norway, 18 December 2007

Irene Johannessen

CONTENTS

CHAPTER

1

INTRODUCTION

A thrust allocation system is used to determine how the desired forces, computed by a high level control sytem, can be distributed among the thrusters. The main goal of the thrust allocation is to obtain the desired force, but other objectives can also be included. Such secondary goals can be to minimize fuel consumption, keep wear and tear of the thruster to a minimum and avoid overloading the power systems. The thrust allocation should also take forbidden sectors and actuator rate constraints into account. It is essential to safe operation that the allocation system provides a solution, and provides the solution in time.

Various strategies using optimization exists to solve the thust allocation problem (Fossen and Johansen,2006). In this thesis the thrust allocation problem will be investigated using a Model Predictive Control (MPC) formulation. MPC is an advanced control technique where an explicitly formulated process model is used to predict and optimize future behaviour. MPC has a great ability to take care of constraints, and this is one of the reasons why it may be well suited to thrust allocation where constraints are always present. Otherwise the hope is that it will minimize fuel consumption.

An online numerical solution will be imlemented, and later on compared with and existing (quasi-)static optimization-based thrust allocation method. This is a method where the angles of the thrusters are held fixed, and only the thrust magnitude can vary for each of the thrusters.

CyberRig I is a scaled model of a semi-submersible drilling unit, and will be

used for testing in real time. The rig has eight rotatable azimuth thrusters and virtual controller specifies surge, sway and yaw forces. Each thruster is limited in thrust magnitude, but can rotate 360 degrees.

Structure of thesis:

In Chapter 2 the existing methods for control allocation are described.

In Chapter 3 a methodological overview of Model Predictive Control is presented.

Chapter 4 describes the problem in detail, MPC formulations are suggested and important assumptions made.

In Chapter 5 results from simulation in Matlab are first presented. Then results from simulations of the online implementation is presented. Results from comparison of MPC control allocation and quasi-static control allocation is shown.

Chapter 6 contains discussion of the results, and a comparation of a MPC formulation and an existing static optimization method is provided.

Chapter 7 contains the conclucion and suggestions for furhter work.

The appendices contain details about hardware, software, lab and some selected code and simulation results.

CHAPTER

2

CONTROL ALLOCATION

Dynamic positioning (DP) is a system that automatically maintain a vessels position and heading by using its propellers and thrusters. On marine vessels under DP operation, control allocation systems are used to determine how the desired generalized forces can be distributed among the thrusters. The control allocator recieves the desired generalized forces from the control system, and computes the neccessary forces and angles for each of the thrusters. A block diagram of the role of the Control Allocation-system in the control hierarchy is shown in Fig. 2.1.

It is critical that the control allocator always provides a solution, and that this solution is made available in time, to provide safe operation.

A vessel can be under-, fully- or overactuated. Underactuated means that the desired generalized force can not be obtained by the available thrusters. On a fully actuated vessel this is possible. However, an overactuated vessel will be the focus in this report. For an overactuated vessel the generalized can force not only be produced, but can be provided in several ways. One then chose a solution based on other criteria, typically in order to minimize power consumption, drag, tear and wear or other costs. Anoter important role of the control allocator is to take into account forbidden sectors and actuator rate constraints.

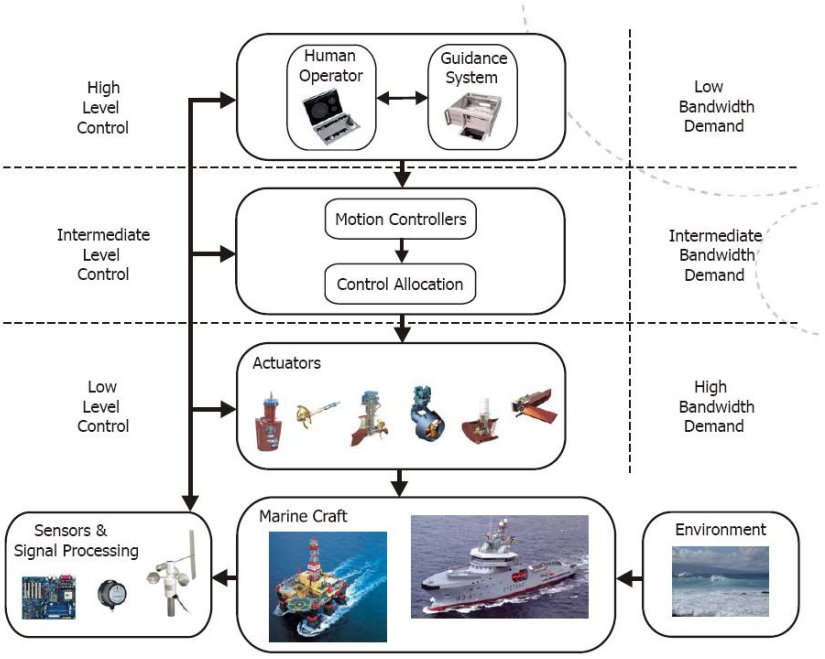


Figure 2.1: Control Hierarchy (Breivik, 2007)

2.0.1 Actuators

Fossen (2002) presents the most common actuators for marine vessels:

Main propeller: Produce the necessary force in the x-direction needed for transit. They are mounted aft of the hull, usually in conjunction with rudders.

Tunnel thrusters: Tranverse thrusters going through the hull of the vessel. The propeller unit is mounted inside a tranverse tube, and it produces force in the y-dirction. Their use is limited to low-speed maneuvering and DP.

Azimuth thrusters: Thruster units that can be rotated about the z-axis, and hence produce two force components in the horizontal plane. They are usually mounted under the hull of the vessel. They are especially well suited for DP systems, due to their ability to produce forces in different direction, leading to optimizable overactuated control problems.

Aft rudders: Primary steering device for conventional vessels, located aft of the vessel. The rudder force F_y will be a function of the rudder deflection. A rudder force in the y-direction will produce a yaw moment which can be used for steering control.

Stabilizing fins: Used for damping of vertical vibrations and roll motions. They produce a force F_z in the z-directions which is a function of the fin deflection. The lift forces are small at low speed, so they work best in transit.

Control surfaces: Produces lift and drag forces and can be mounted at different locations.

2.1 Modelling

Following Fossen (2002) the forces and moments in 6 DOF corresponding to the force vector $\mathbf{f} = [F_x, F_y, F_z]^T$ can be written:

$$\tau = \begin{bmatrix} \mathbf{f} \\ \mathbf{r} \times \mathbf{f} \end{bmatrix} = \begin{bmatrix} F_x \\ F_y \\ F_z \\ F_z l_y - F_y l_z \\ F_y l_z - F_z l_x \\ F_y l_x - F_x l_y \end{bmatrix} \tag{2.1}$$

Forces and moments can be written:

$$\tau = \mathbf{T}(\alpha)\,\mathbf{f} \tag{2.2}$$

$$\mathbf{f} = \mathbf{K}\mathbf{u} \tag{2.3}$$

where α are the angles (in the case of azimuth thruster) and $\mathbf{u}$ are the control inputs. $\mathbf{f} = \mathbf{K}\mathbf{u}$ is a vector of control forces. An azimuth thruster have two force components $F_x = Fcos\alpha$ and $F_y = Fsin\alpha$.

$\mathbf{K}$ is called the *force coefficient matrix* and is always diagonal:

$$\mathbf{K} = diag\{K_1, K_2, \cdots, K_I\} \tag{2.4}$$

$$\mathbf{K}^{-1} = diag\left\{\frac{1}{K_1}, \frac{1}{K_2}, \cdots, \frac{1}{K_I}\right\} \tag{2.5}$$

where I is the number of actuators.

$\mathbf{T}(\alpha)$ is called the *actuator configuration matrix*. It is written:

$$\mathbf{T}(\alpha) = [t_1, t_2, \ldots, t_I] \tag{2.6}$$

The column vector $\mathbf{t}_i$ for an azimuth thruster with 5 DOF is:

$$\mathbf{t}_i = \begin{bmatrix} \cos\alpha_i \\ \sin\alpha_i \\ -l_{zi}\sin\alpha_i \\ l_{zi}\cos\alpha_i \\ l_{xi}\sin\alpha_i - l_{yi}\cos\alpha_i \end{bmatrix} \tag{2.7}$$

Dynamic positioning is concerned primarily with control of the ship in the horizontal plane: surge, sway and yaw.3 DOF representation (with surge, sway and yaw) is found by deleting the 3rd and 4th row, and $\mathbf{t}_i$ becomes:

$$\mathbf{t}_i = \begin{bmatrix} \cos\alpha_i \\ \sin\alpha_i \\ l_{xi}\sin\alpha_i - l_{yi}\cos\alpha_i \end{bmatrix} \tag{2.8}$$

In low-speed allocation it is common to use a 3-degrees-of-freedom model, taking surge, sway and yaw into account. The generalized forces (vessel fixed) is then $\tau = [X, Y, N]^T$.

The generalized forces acting on the vessel is given by:

$$X_i = T_i \cos\alpha_i, \tag{2.9}$$

$$Y_i = T_i \sin\alpha_i, \tag{2.10}$$

$$N_i = T_i(l_{i,x}\sin\alpha_i - l_{i,y}\cos\alpha_i) \tag{2.11}$$

2.2 Control allocation methods

A good survey of methods for control allocation is presented by Fossen and Johansen (2006). They divide the methods into three categories:

- Linear quadratic unconstrained control allocation
- Linear quadratic constrained control allocation
- Nonlinear constrained control allocation

2.2.1 Linear quadratic unconstrained contol allocation

Linear quadratic unconstrained control allocation is the simplest form of allocation problem. The easiest allocation problem is obtained if the thrusters do not rotate: the angle α is a constant α_0 and $T(\alpha) = T(\alpha_0) = constant$. If the allocation in addition is *unconstrained* the problem is simple to solve. If it is equal or more control inputs than controllable DOF, it is possible to

find a ('optimal') solution using an explicit method. Considering an unconstrained least-squares optimization problem one can compute a *generalized inverse* $\mathbf{T}_w^\dagger$ and then find the control input vector $\mathbf{u}$ as:

$$\mathbf{u} = \mathbf{K}^{-1}\mathbf{T}_w^\dagger \boldsymbol{\tau} \tag{2.12}$$

The LS optimization problem in question is given by:

$$\begin{aligned} J &= \min_{\mathbf{f}} \{\mathbf{f}^T\mathbf{W}\mathbf{f}\} \\ &\quad subject\ to: \boldsymbol{\tau} - \mathbf{T}\mathbf{f} = \mathbf{0} \end{aligned} \tag{2.13}$$

$\mathbf{W}$ is a matrix (positive definite) weighting the control forces. The generalized matrix can then be found as:

$$\mathbf{T}_w^\dagger = \mathbf{W}^{-1}\mathbf{T}^T\left(\mathbf{T}\mathbf{W}^{-1}\mathbf{T}^T\right)^{-1} \tag{2.14}$$

If the thrusters are allowed to rotate, one can solve this problem using an *extended thrust representation* (Sørdalen, 1997). Following the notation in Spjøtvold and Johansen(2007), the extended thrust vector is found by decomposing the individual thrust vectors in the horizontal plane according to: $u_{i,x} := X_i,\ u_{i,y} = Y_i$ and $u_i := [u_{i,x}\ u_{i,y}]^T \in R^2$.

This way a linear relationship is obtained:

$$\tau = Bu \tag{2.15}$$

u is the decomposed thust vectors in the horizontal plane for all the actuators: $u = [u_{1,x}, u_{1,y}, u_{2,x}, u_{2,y}\ \cdots\ u_{I,x}, u_{I,y},]^T$, and the matrix B is:

$$B = \begin{bmatrix} 1 & 0 & \cdots & 1 & 0 \\ 0 & 1 & \cdots & 0 & 1 \\ -l_{1,y} & l_{1,x} & \cdots & -l_{I,y} & l_{I,x} \end{bmatrix} \tag{2.16}$$

The thrust force T_i can be found as $\sqrt{u_x^2 + u_y^2}$ and the angle α_i are recovered by $arctan(\frac{u_y}{u_x})$.

2.2.2 Linear quadratic constrained control allocation

The solutions in Sec. 2.2.1 does not take constraints into account. Considering actuator limitations such as saturation and tear and wear leads to a constrained allocation problem. Other constraints can be due to the wish

to minimize the fuel consumption and hence the costs. Various approaches exists to solve the constrained control allocation problem, and some will be shortly described.

Firstly, an explicit solution using piecewise linear functions is presented. Tøndel et al.(2003) developed an explicit solution approach for parametric quadratic programming, and Johansen et al. (2005) presented applications to marine vessels. The constrained opimization problem is given as:

$$\begin{aligned} J &= \min_{\mathbf{f},\mathbf{s},\bar{f}} \left\{ \mathbf{f}^T \mathbf{W} \mathbf{f} + \mathbf{s}^T \mathbf{Q} \mathbf{s} + \beta \bar{f} \right\} \\ &\quad subject\ to: \\ &\quad \mathbf{T}\mathbf{f} = \boldsymbol{\tau} - \mathbf{s} \\ &\quad \mathbf{f}_{min} \leq \mathbf{f} \leq \mathbf{f}_{max} \\ &\quad -\overline{f} \leq f_1, f_2, \ldots, f_r \leq \overline{f} \end{aligned} \tag{2.17}$$

where $\mathbf{s}$ is a vector of *slack variables*. Choosing the weighting matrix $\mathbf{Q} \succ\succ \mathbf{W}$ makes the slack variables close to zero, and assures an accurate generalized force. The optimization problem can be reformulated as a convex QP problem with $\mathbf{p} = \left[\boldsymbol{\tau}^T, \mathbf{f}_{min}^T, \mathbf{f}_{max}^T, \beta\right]^T$ as the parameter vector and $\mathbf{z} = \left[\mathbf{f}^T, \mathbf{s}^T, \bar{f}\right]^T$. The QP problem is a convex quadratic program in $\mathbf{z}$ parametrized by $\mathbf{p}$, where a global solution is found due to covexity. The optimal solution $\mathbf{z}^*(\mathbf{p})$ is a continuous piecewise linear function. An exact representation can be computed offline using mpQP-algorithms. The reader should go to the references for a more throughout explanation.

The above method applies to a vessel with nonrotatable actuators. An extension to vessels with azimuthing thrusters has also been presented (Johansen et al, 2003).

Other solutions to the constrained control allocation problem are explicit solutions based on Minimum Norm and Null-space methods. They solve the control allocation problem applied in flight and aerospace control systems. Also another possibility is mentioned: to use an iterative solution to solve the QP-problem. This method is more flexible, but may require that several iterations are performed at each sample in order to find the optimal solution.

2.2.3 Nonlinear constrained control allocation

A vessel equipped with azimuth thrusters leads to a non-convex optimization problem that is hard to solve. A possible criterion for minimization is presented in Johansen et al(2004b):

$$
\begin{aligned}
J = \min_{\mathbf{f},\mathbf{s},\boldsymbol{\alpha}} & \sum_{i=1}^{r} \overline{P}_i \left|f_i\right|^{3/2} + \mathbf{s}^T \mathbf{Q} \mathbf{s} \\
& + (\boldsymbol{\alpha} - \boldsymbol{\alpha}_0)^T \boldsymbol{\Omega} (\boldsymbol{\alpha} - \boldsymbol{\alpha}_0) \\
& + \frac{\varrho}{\varepsilon + det(\mathbf{T}(\boldsymbol{\alpha}) \mathbf{W}^{-1} \mathbf{T}^T(\boldsymbol{\alpha}))}. \\
subject\ to: & \\
& \mathbf{T}(\boldsymbol{\alpha}) \mathbf{f} = \boldsymbol{\tau} + \mathbf{s} \\
& \mathbf{f}_{min} \leq \mathbf{f} \leq \mathbf{f}_{max} \\
& \boldsymbol{\alpha}_{min} \leq \boldsymbol{\alpha} \leq \boldsymbol{\alpha}_{max} \\
& \Delta\boldsymbol{\alpha}_{min} \leq \boldsymbol{\alpha} - \boldsymbol{\alpha}_0 \leq \Delta\boldsymbol{\alpha}_{max}
\end{aligned}
$$

The first term in J represents power consumption, and the last term is included to avoid the singularity that is introduced if $det(T(\alpha)W^{-1}T^T(\alpha))$ is equal to zero. The resulting problem is a non-convex nonlinear program. Therefore Fossen and Johansen (2006) suggests two implementation strategies in order to avoid the demanding computations.

Firstly, one can find a dynamic solution using Lyapunov methods. For more information about this, the reader is refered to Johansen (2004). Secondly, an iterative solution using quadratic programming is proposed. This is done approximation the problem in Eq. 2.18 with a convex QP problem that is easier to solve. (Johansen et al., 2004b).

Two other methods to solve the problem in Eq. 2.12 are also mentioned: Iterative solutions using linear programming, and explicit solution using the singular value decomposition and filtering techniques.

2.3 Thruster model

It is assumed that the thrust force from each thruster is given by (Fossen, 2002):

$$T = K_T \rho D^4 \left|n\right| n \tag{2.18}$$

K_T is a strictly positive thrust coefficient, ρ is the water density, D is the propeller diameter and n is the propeller speed. ρ and K_T are assumed to be constant, and the propeller speed n is recovered from:

$$n = sgn(T) * \sqrt{\left|\frac{1}{\rho K_T D^4}\right|} * 60 \tag{2.19}$$

In the equation 60 is included to transform from rps (revolutions per second) to rpm (revolutions per minute).

2.4 Thruster-thruster Interactions

When two azimuthing thrusters are operating within close prximity to each other, interaction between their respective slipstreams can occur depending on their azimuth angles α. For the rig this is highly relevant, as the thrusters are mounted in pairs on each leg of the rig. Brown and Ekstrom (2005) states that either a reduction or an increase in thrust can occur when one thruster's slipstream is influenced by the other thruster's slipstream. Little litterature exists on the topic, and experimental tests should be conducted. Analytical results are difficult to derive due to the complexity of hydrodynamical flow.

CHAPTER

3

MODEL PREDICTIVE CONTROL

Model predictive control (MPC) is one of the most commonly used techniques within advanced control today. Its first applications were in the petrochemical industry, but due to several factors it is gaining increasingly popularity also in other areas. An important reason for this is its ability to take care of constraints (Maciejowski, 2002). There are many variants of MPC controllers, having in common that an explicitly formulated process model is used to predict and optimize future behaviour.

In the literature the term receding horizon control (RHC) is occasionally used. MPC is the conventional name of this technique, and is the term being refered to in this report.

The purpose of this chapter is to give a short survey of the history of MPC and its theory.

Mayne, Rawlings, Rao and Scokaert (2002) says that:

> "Model predictive control is a form of control in which the current control action is obtained by solving, at each sampling instant, a finite horizon open-loop optimal control problem, using the current state of the plant as the initial state; the optimization yields an optimal control sequence and the first control in this sequence is applied to the plant."

The open-loop control sequence is repeatedly solved at each time step, giving the the controller an inherently closed-loop effect. As the new control sequence is calculated from the present state of the system, a stabilizing feedback control can be obtained. This is its main difference from conventional control which uses a pre-computed control law. The ability to handle control problems where off-line computation of the control law is difficult or impossible is one of the greatest advantages by using this technique. However, initially other features suchs as capability for controlling multivariable plants were considered more important (Mayne et.al, 2002).

The main reason why MPC first gained popularity was its ability to handle constraints. Constraints, such as actuators beeing limited in the force they can apply and safety limits in temperature, pressure and velocity, are almost always present. Steady-state operation close to the boundary of the set of permissible states leads to more profitable operation.

The time consuming on-line computations initally led to MPC beeing used only in slow processes. However, with faster computing hardware and improved optimization algorithms, there is no need for predictive control beeing confined only to slow processes (Maciejowski, 2002).

3.1 The idea behind receding horizon

Fig. 3.1 from Torpe(2007) shows the principle of MPC. A system is sought to be controlled at a set point, given by r(t). The controller calculates an optimal input sequence for the control horizon T_c. The first element of the input sequence is applied as the input signal to the plant. As the time progresses all horizons are moved ahead as well, so they slide along by one sampling interval at each step.

A mathematical formulation is now given, following Allgöver, Findeisen and Nagy (2004). A continuous time formulation is provided, se for example Mayne et.al(2000) for a discrete time formulation. Consider a general class of continious time systems described by a differencial equation

$$\dot{x}(t) = f(x(t), u(t)), \quad x(0) = x_0, \tag{3.1}$$

which is subject to input and state constraints of the form:

$$u(t) \in U, \ \forall t \geq 0, \tag{3.2}$$
$$x(t) \in X, \ \forall t \geq 0. \tag{3.3}$$

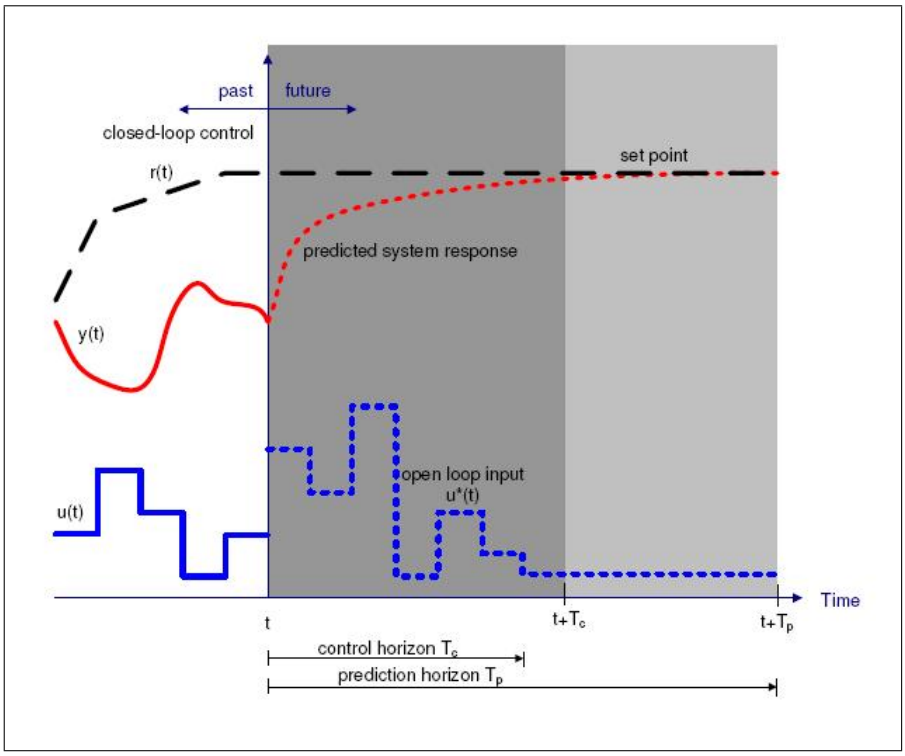


Figure 3.1: Principle of MPC

Here $u(t) \in \boldsymbol{R}^m$ is the vector of inputs, and $x(t) \in \boldsymbol{R}^n$ is the state vector. The sets U and X typically takes the form:

$$U := u \in R^m \, | u_{min} \leq u \leq u_{max}, \tag{3.4}$$

$$X := x \in R^n \, | x_{min} \leq x \leq x_{max}, \tag{3.5}$$

with u_{min}, u_{max}, x_{min} and x_{max} beeing constant vectors.

At every time instant the optimal open-loop control is given by solving the following problem:

$$\underset{\overline{u}(\cdot)}{min} J(x(t), \overline{u}(\cdot)) = \underset{\overline{u}(\cdot)}{min} \int_t^{t+T_p} F(\overline{x}(\tau), \overline{u}(\tau)) d\tau \tag{3.6}$$

subject to

$$\dot{\overline{x}} = f(\overline{x}(\tau), \overline{u}(\tau)), \overline{x}(t) = x(t), \tag{3.7}$$

$$\overline{u}(\tau), \forall \tau \in [t, t + T_c] \, , \tag{3.8}$$

$$\overline{u}(\tau) = \overline{u}(t + T_c), \forall \tau \in [t + T_c, t + T_c] \, , \tag{3.9}$$

$$\overline{x}(\tau) \in X, \forall \tau \in [t, t + T_p] \tag{3.10}$$

T_p is the prediction horizon and T_c the control horizon. $\overline{u}$ denotes internal controller variables, while $\overline{x}$ denotes the system response to the input vector $\overline{u}$, that is, the solutions to Eq. 3.6. The cost function J is as sum of performance costs, $F(\cdot)$ at each time step. The cost function often arises from some economical consideration on the systems operational point (x_{op}, u_{op}) through a quadratic form:

$$F(x,u) = (x - x_{op})^T Q(x - x_{op}) + (u - u_{op})^T R(u - u_{op}) \tag{3.11}$$

That is, the cost is given as a result of deviations form the operational set point, specified by positive definite *weighting matrices* Q and R. One can also find additional terms in F which penalize movements of the inputs where that is appropriate.

The solution to Eq. 3.6 is defined to be $\overline{u}^*(t_0; x_0) = \overline{u}_0^*$, where the first set of inputs are applied to the system:

$$u(t) = \overline{u}^*(t_0, x_0) = \overline{u}_0^*. \tag{3.12}$$

The opitmal cost yielded by $\overline{u}^*$ is then a function of the stat $x(t)$ alone. This optimal cost is often referred to as the value function

$$V(x) = J(x(t), \overline{u}^*(t; x(t))). \tag{3.13}$$

3.2 Origin of MPC

For a while, there where several quite independent streams in the MPC litterature: the one dealing with theoretical foundations, the 'process control litterature' used by the indystry, and litterature on generalized predictive control having its roots in minimum variance and adaptive control (Mayne et.al, 2002).

Theoretical foundations: Optimal control litterature is highly relevant to the development op MPC. Firstly two ideas in the optimal control litterature from the 1960s underly MPC: *Hamilton-Jacobi-Bellman therory* (Dynamic Programming) and the *maximum principle*. DP provides sufficient contidions for optimality and a method to find an optimal feedback controller $u = \kappa(x)$. The maximum principle motivates computational algorithms to, given an initial state x, determinate the optimal open-loop control $u^0(\cdot; x)$. "The link is: $\kappa(x) = u^0(0; x)$ for all x", meaning that optimal feedback can be obtained by solving an open-loop control problem for each state x.

Secondly there is the important observation by Kalman (1960), saying that optimality does not imply stability. However, infinite horizon optimal controllers are stabilizing under given circumstances. The controller is known as the Linear Quadratic Regulator (LQR) (Kalman, 1960a,b).

The process control litterature: LQG theory (Linear quadratic Gaussian) soon gained popularity in a wide range of applications. But this did not affect the control technology developement in the process industries much (Qin and Badgewell, 2003). The theory did not correspond to many of the concerns of the industry, beeing:

- constraints
- process nonlinearities
- model uncertainty (robustness)
- unique performance criteria
- cultural reasons (people, education, etc.)

Stability were not addressed by the industrial proponents of MPC, and the controllers were not automatically stabilizing. But the focus were on stable plants and using large horizon, hence stability properties are achieved.

Generalized predictive control: Generalized predictive control has its roots in adaptive control litterature. Stability was not guaranteed in the original versions of GPC, due to finite horizon, but was achieved by tuning (of cost and horizon parameters). In spite of its possible potencial in the market for a self-tuning MPC controller, very few adaptive MPC algorithms have reached the marketplace. Qin and Badgewell (2003) states that this as a situacion that is unlikely to change unless there is a theoretical breakthrough.

This concludes the chapter on MPC theory. From a focus on methodological overview, the following chapter suggests to use MPC in control allocation, and more specifically, applied on CyberRig I.

CHAPTER

4

PROBLEM DESCRIPTION AND IMPLEMENTATION

As seen in Chapter 2 control allocation systems on overactuated vessels are important in order to obtain the required generalized thrust and minimize power consumption. In this chapter the control allocation problem for a marine vessel is proposed solved using a MPC formulation. A few formulations will be suggested, and the most promising will be chosen for implementation on a real-life model of semi-submersible drilling unit. Importent aspects on implementation, such as how to implement constraints, will also be commented on.

4.1 CyberRig I

As a case study CyberRig I will be used. It is a scaled model of a semi-submersible drilling unit. The model has four legs with two azimuth thrusters on each leg. The CyberRig I (CRI) has 8 rotating thrusters, giving many degrees of fredom and leading to a problem that is higly optimizable. Implemented on CRI is a DP control system (Tyssø and Aga, 2006). The CyberRig I is shown in Fig. 4.1, while the numbering of the thrusters is shown in Fig. 4.2.

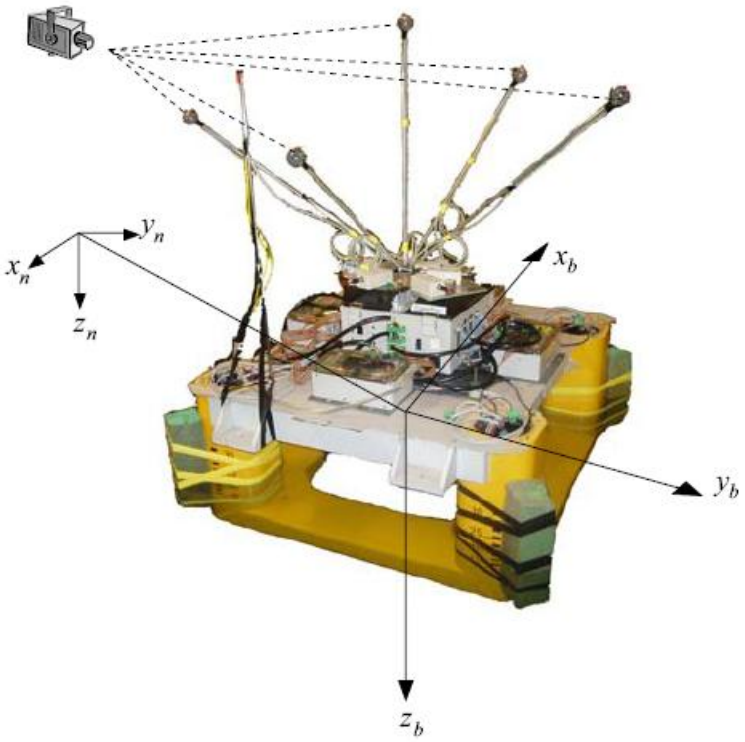


Figure 4.1: CyberRig I with camera positioning system

4.2 MPC-formulations

To solve the control allocation problem 3 MPC-formulations will be suggested. They will all be simulated in Matlab, and one will be chosen for implementation on CyberRig I. To include the varying angles, the work will be based on the *extended thrust* representation described in Sec. 2.2.1. Traditionally this gives the problem that the optimal solution for α can jump at each sample, and that constraints can not be implemented. However, by combining this representation with MPC this can be managed. With MPC, constraints are easily implemented. For control allocation both rate constraints and forbidden sectors should be included, and with MPC they can be. MPC is also well suited because it handles error situations. If one of the thrusters fail, the associated control can be set to zero in the equality constraints (or removed from the formulation), and the other constraints will take care of optaining the commanded force.

First of all the notation will be decribed.

The thrust vector is given by:

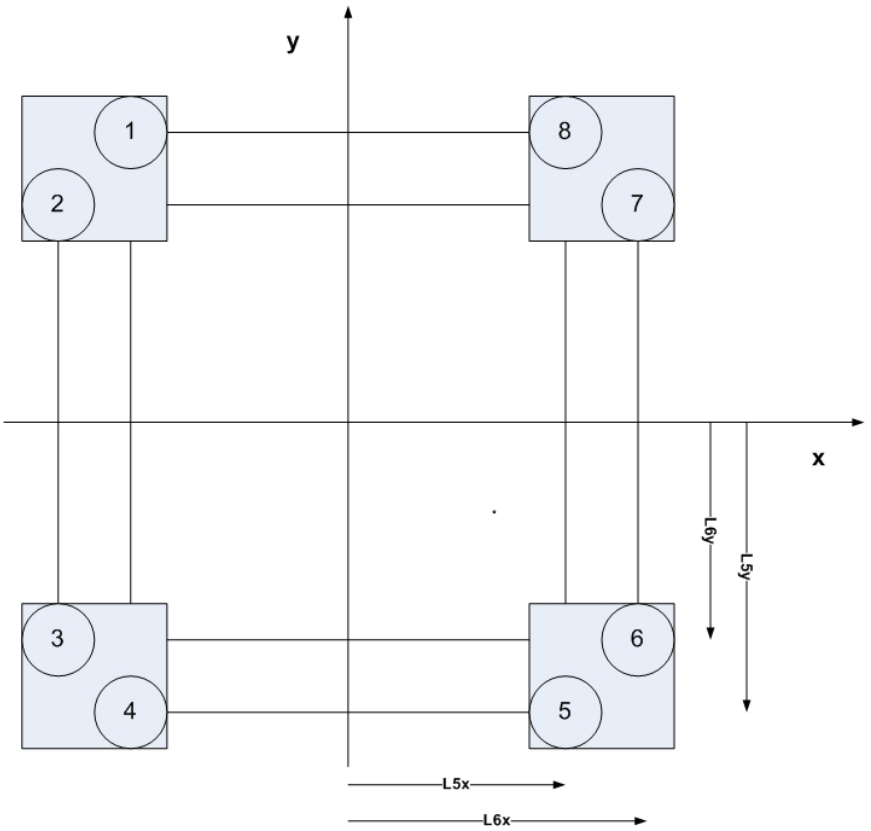


Figure 4.2: Numbering of azimuth thrusters

$$u = u_i^k \tag{4.1}$$

k is the time index, $k \in [1, \ldots, N]$ and i is the index of the azimuth thruster $i \in [1, \ldots, I]$. N is the time horizon used for the prediction algorithm, and $I = 8$ azimuth thrusters for CyberRig I. Also remember from Sec. 2.2.1 that $u_i = [u_{i,x}\ u_{i,y}]^T$.

α is the thusters angle, and T the force magnitude.

$$\alpha_i = arctan\left(\frac{u_{i,y}}{u_{i,x}}\right) \tag{4.2}$$

$$T_i = \sqrt{u_{i,x}^2 + u_{i,y}^2} \tag{4.3}$$

$\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_I]$ and $T = [T_1, T_2, \ldots, T_I]$.

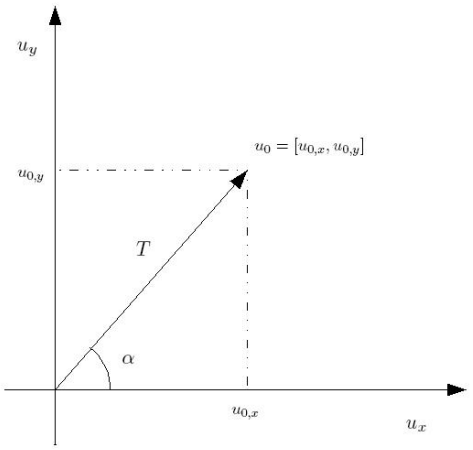


Figure 4.3: T and alpha

Algorithm 1:

In the first algorithm it will be assumed that α^* and T^* are the 'optimal' values for α and T for the present commanded force τ_{comm}. They will be computed by an optimization algorithm that also takes forbidden sectors into account. This optimization algorithm is a QP-problem minimizing the force u, with $\tau_{comm} = Bu + s$ as equality constraints and forbidden sectors as unequality constraints.

$$\begin{aligned} J \quad = \quad & \Sigma_{k=1}^{N}((\alpha^*)^k - \alpha^k)^T Q((\alpha^*)^k - \alpha^k) + ((T^*)^k)^T R(T^k) \\ & subject\ to: \\ & A_1^k \alpha^k \leq b_1^k, \quad \forall\ k \in [1, N] \\ & A_2^k T^k \leq b_2^k, \quad \forall\ k \in [1, N] \\ & -\Delta\alpha_{min}^k \leq \Delta\alpha^k \leq \Delta\alpha_{max}^k, \quad \forall\ k \in [1, N] \\ & -\Delta T_{min}^k \leq \Delta T^k \leq \Delta T_{max}^k, \quad \forall\ k \in [1, N] \end{aligned} \tag{4.4}$$

This algorithm has the advantage that it deals with the problem of constraints, as the rate constraints can be easily implemented.

Algorithm 2:

For algorithm number 2 the goal is to minimize the error between the desired and implemented generalized force over the time horizon. The cost function J is formulated as:

$$
\begin{aligned}
J &= \Sigma_{k=1}^{N} \left\| (Bu^k - \tau_{comm}^k) \right\|_{Q^k} \\
& \quad subject\ to: \qquad (4.5) \\
& \quad A^k u^k \leq b_1^k, \ \ \forall\ k \in [1, N] \qquad (4.6) \\
& \quad -\Delta\alpha_{min}^k \leq \Delta\alpha^k \leq \Delta\alpha_{max}^k, \ \ \forall\ k \in [1, N] \\
& \quad -\Delta T_{min}^k \leq \Delta T^k \leq \Delta T_{max}^k, \ \ \forall\ k \in [1, N]
\end{aligned}
$$

It is more complicated to implement because the varibles for rate constraints are not the same as the optimization vector (see Sec. 4.3.2). But it may take greater advantage of the prediction horizon.

Algorithm 3:

The MPC algorithm should also minimize power consumption. To do this it is normal to add an energy term to the cost function. This can be approximated to be u^2 for simplicity. But with the formulation in Eq. 4.7 this is not possible, as it violates the minimisation of the error term.

To overcome this problem a third algorithm is suggested:

$$
\begin{aligned}
J &= \Sigma_{k=1}^{N} \|(u^k - u^{k-1})\|_Q + \|s^k\|_R \qquad (4.7) \\
& \quad subject\ to: \qquad (4.8) \\
& \quad Bu^k + \boldsymbol{s}^k = \boldsymbol{\tau}^k, \ \ \forall\ k \in [1, N] \qquad (4.9) \\
& \quad A^k u^k \leq b^k, \ \ \forall\ k \in [1, N] \qquad (4.10) \\
& \quad -\Delta\alpha_{min}^k \leq \Delta\alpha^k \leq \Delta\alpha_{max}^k, \ \ \forall\ k \in [1, N] \qquad (4.11) \\
& \quad -\Delta T_{min}^k \leq \Delta T^k \leq \Delta T_{max}^k, \ \ \forall\ k \in [1, N] \qquad (4.12)
\end{aligned}
$$

This algorithm includes the power consumption in the first term of Eq. 4.7, and also weight that change in angle also influence on the power consumption. Following of the reference $\boldsymbol{\tau}_{comm}$ is assured by the first equality constraint. $\mathbf{s}$ is a vector of slack variables, allowing the following of the reference to be violated if this is neccesary due to the other contstraints. But the weighting matrix R is chosen to be much larger than Q to make $\mathbf{s}$ as small as possible.

Please notice that also the constraints are varying at each sample for the above formulations. In the following sector some implementation issues are commented on.

4.2.1 Implementation

In order to implement the mpc algorithms available software must be used. Therefore the problems are formulated as QP problems that have solvers both in matlab and in C available (quadprog and qld respectively, see App. C).

To do this the problems for each time istant is simply stacked. In this section it will be shown how the stacking is done for algorithm 3, although the procedure is the same for all of the algorithms.

$$\mathbb{U} = \begin{bmatrix} u^1 \\ u^2 \\ \vdots \\ u^N \end{bmatrix} \tag{4.13}$$

$$u^1 = \begin{bmatrix} u^1_{1,x} \\ u^1_{1,y} \\ \vdots \\ u^1_{I,x} \\ u^1_{I,y} \end{bmatrix}, \quad u^2 = \begin{bmatrix} u^2_{1,x} \\ u^2_{1,y} \\ \vdots \\ u^2_{I,x} \\ u^2_{I,y} \end{bmatrix} \quad etc. \tag{4.14}$$

$$\mathbb{S} = \begin{bmatrix} s^1 \\ s^2 \\ \vdots \\ s^N \end{bmatrix} \tag{4.15}$$

$$s^1 = \begin{bmatrix} s^1_1 \\ s^1_2 \\ s^1_3 \end{bmatrix}, \quad s^2 = \begin{bmatrix} s^2_1 \\ s^2_2 \\ s^2_3 \end{bmatrix} \quad etc. \tag{4.16}$$

N is the time horizon and I is the number of actuators.

$$\begin{aligned} J &= \Sigma_{k=1}^{N} \|(u^k - u^{k-1})\|_Q + \|s^k\|_R && (4.17)\\ &= \mathbb{U}^T\mathbb{Q}\mathbb{U} - \mathbb{U}^T\mathbb{Q}\mathbb{U}^0 - \mathbb{U}\mathbb{Q}(\mathbb{U}^0)^T + (\mathbb{U}^0)^2 + \mathbb{S}\mathbb{R}\mathbb{S}^T && (4.18)\\ &= \mathbb{U}^T\mathbb{Q}\mathbb{U} - 2(\mathbb{U}^0)^T\mathbb{Q}\mathbb{U} + (\mathbb{U}^0)^2 + \mathbb{S}\mathbb{R}\mathbb{S}^T && (4.19) \end{aligned}$$

$$Q^k = \underbrace{\begin{bmatrix} q & & & 0 \\ & q & & \\ & & \ddots & \\ 0 & & & q \end{bmatrix}}_{I}, \quad R^k = \underbrace{\begin{bmatrix} r & & 0 \\ & r & \\ 0 & & r \end{bmatrix}}_{3} \tag{4.20}$$

$$\mathbb{Q} = \underbrace{\begin{bmatrix} Q^1 & & & 0 \\ & Q^2 & & \\ & & \ddots & \\ 0 & & & Q^N \end{bmatrix}}_{N}, \quad \mathbb{R} = \underbrace{\begin{bmatrix} R^1 & & 0 \\ & R^2 & & \\ & & \ddots & \\ 0 & & & R^N \end{bmatrix}}_{N} \tag{4.21}$$

This results in the vector to optimize over:

$$\mathbb{X} = \begin{bmatrix} \mathbb{U} \\ \mathbb{S} \end{bmatrix} = \begin{bmatrix} u^1 \\ u^2 \\ \vdots \\ u^N \\ s^1 \\ \vdots \\ s^N \end{bmatrix} \tag{4.22}$$

$$J = \min_{\mathbb{X}} \frac{1}{2} \mathbb{X}^T H \mathbb{X} + f^T \mathbb{X} \tag{4.23}$$

$$H = 2 \begin{bmatrix} \mathbb{Q} & 0 \\ 0 & \mathbb{R} \end{bmatrix}, \quad f = 2(\mathbb{X}^0)^T H_2, \quad H_2 = 2 \begin{bmatrix} \mathbb{Q} & 0 \\ 0 & \mathbb{O} \end{bmatrix} \tag{4.24}$$

$\mathbb{X}^0$ is the optimal $\mathbb{X}^*$ from the last step.

The constraints must be on the form $\mathbb{A}_{uneq}\mathbb{X} \leq \mathbb{B}_{uneq}$ and $\mathbb{A}_{eq}\mathbb{X} = \mathbb{B}_{eq}$.

The equality constraints takes the form:

$$\underbrace{\begin{bmatrix} B^1 & & & I & & \\ & B^2 & & & I & \\ & & \ddots & & & \ddots \\ & & & B^N & & I \end{bmatrix}}_{\mathbb{A}_{eq}} \mathbb{X} = \underbrace{\begin{bmatrix} \tau^1 \\ \tau^2 \\ \vdots \\ \tau^N \end{bmatrix}}_{\mathbb{B}_{eq}} \tag{4.25}$$

where

$$I = \begin{bmatrix} 1 & & \\ & 1 & \\ & & 1 \end{bmatrix}, \quad B^i = B \ \ \forall k \in [1, N] \tag{4.26}$$

and B is the linearized matrix from Eq. 2.16. It will be assumed that the future commanded forces τ^k are equal to the present one, τ_{comm}.

When it comes to the unequality constraints two types are implemented: forbidden sector constraints, and rate constraints. The rate constraints are on the form $\Delta\alpha$ and ΔT, but should be on the form $Au \leq b$. They will be closer described in Sec. 4.3.2.

The forbidden sector constraints will be on the form:

$$\underbrace{\begin{bmatrix} sa_1 & & & \\ & sa_2 & & \\ & & \ddots & \\ & & & sa_I \end{bmatrix}}_{SA} u \leq \underbrace{\begin{bmatrix} sb_1 \\ sb_2 \\ \vdots \\ sb_I \end{bmatrix}}_{SB} \tag{4.27}$$

sa_i are matrices and sb_i are vectors. The forbidden sector constraints do not vary from sample to sample, and can be simply stacked to acheive the constraints for the whole prediction horizon:

$$\begin{bmatrix} SA^1 & & & 0 & & \\ & SA^2 & & & 0 & \\ & & \ddots & & & \ddots \\ & & SA^N & & & 0 \end{bmatrix} \mathbb{X} \leq \begin{bmatrix} SB^1 \\ SB^2 \\ \vdots \\ SB^N \end{bmatrix} \tag{4.28}$$

The forbidden sector constraints is described more closely in Sec. 4.3.1.

The result of the optimization is $\mathbb{X}^*$, containing $\mathbb{U}^*$ beeing the future vector of forces. The first term $\mathbb{U}^{1,*}$ is applied to the plant, and the optimization algorithm is redone. Fig. 4.4 shows a blockdiagram of the process.

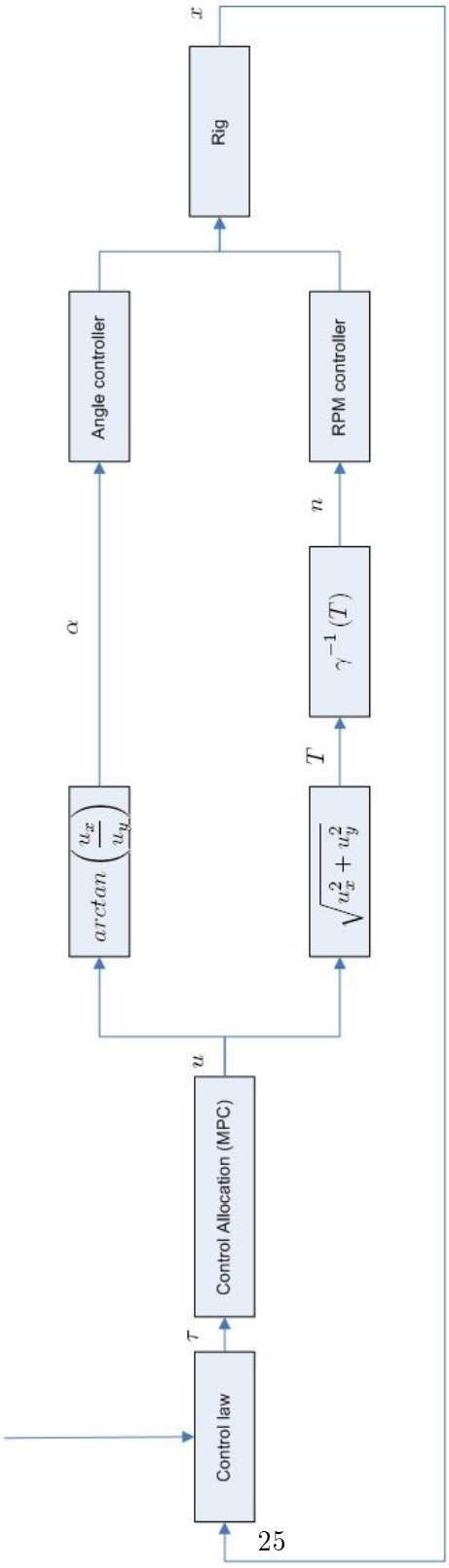


Figure 4.4: Block diagram

4.3 Constraints

4.3.1 Forbidden sectors

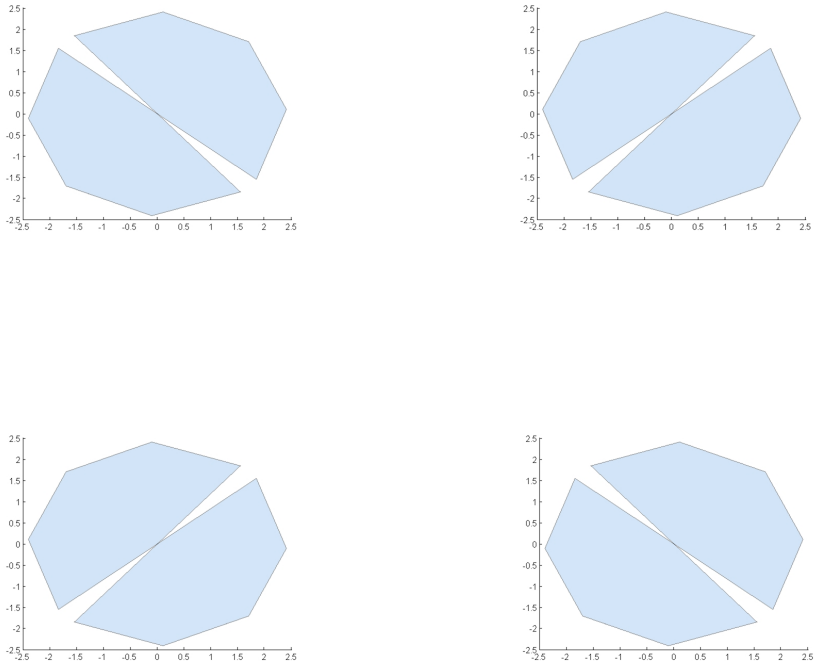


Figure 4.5: Allowed sectors for the thrusters

Forbidden sectors are introduced to prevent the thrusters to enter certain sectors. The sectors are artificial in the sense that they are introduced to avoid non-linear interaction between the thrusters. Forbidden sectors are also useful to prevent the thruster to pump water in certain directions due to for example maintenance (boats, divers etc. in the sea).

In implementation instead of forbidden sectors, constraints are made to keep each thruster within a certain area. This way minimum and maximum values for T and α are also implemented. In Fig. 4.5 the shadowed area is the allowed area for the thrusters. The allowed sectors are approximations of half circles. The half circles are approximated by the half of regular polygons.[1]

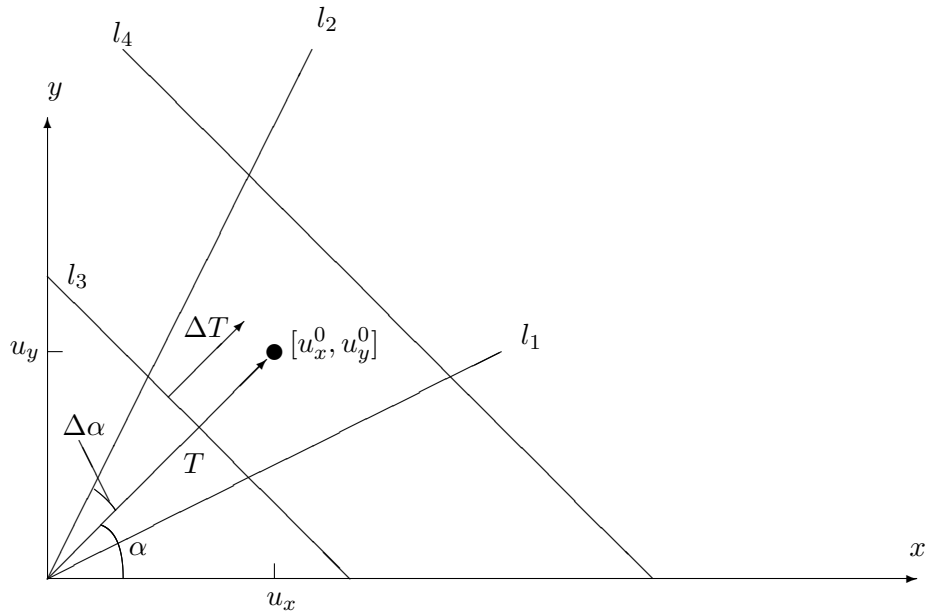


Figure 4.6: Rate constraints

4.3.2 Rate constrains

From step to step it is desirable that the rpm and angles of the azimuth thrusters only can change by certain values. Therefore rate constraints are introduced. In Fig. 4.6 these constraints are visualized by a polynomial. The allowed section for the next step limited by the lines l_1, l_2, l_3 and l_4 on the figure.

For a given point $p = [u_x^0, u_y^0]$ the angle α and magintude T are given, see Fig. 4.6. For the next iteration $\alpha_{max} = \alpha + \Delta\alpha$ and $\alpha_{min} = \alpha - \Delta\alpha$. For the maginitude equally: $T_{max} = T + \Delta T$ and $T_{min} = T - \Delta T$.

But to be implemented in the qp-problem, the constraints should be on the form $Au^1 \leq b$. ($u^1 = [u_x^1, u_y^1]$).

Let p_1 be a point on line l_1 and p_2 be a point on line l_2 such that:

$$p_1 = [p_{1,x}, p_{1,y}] = [cos(\alpha_{min}),\ sin(\alpha_{min})] \tag{4.29}$$
$$p_2 = [p_{2,x}, p_{2,y}] = [cos(\alpha_{max}),\ sin(\alpha_{max})] \tag{4.30}$$

The constraints for alpha are then:

$$\begin{bmatrix} p_{1,y} & -p_{1,x} \\ -p_{2,y} & p_{2,x} \end{bmatrix} \begin{bmatrix} u_x^1 \\ u_y^1 \end{bmatrix} \leqq \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.31}$$

[1] A regular polygon is a polygon which is equiangular (all angles are congruent) and equilateral (all sides have the same length).

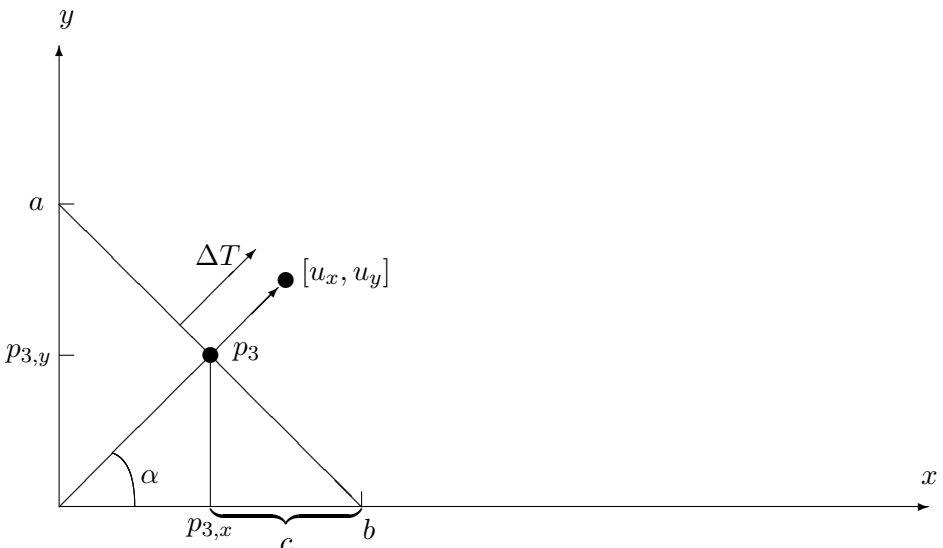


Figure 4.7: Illustration rate constraints, magnitude

The magnitude constraints are constructed as follows:

The technique is different depending on the quadrant, here an example of how to find the constraint for line l_3 in the first quadrant is given.

Let p_3 be a point on line l_3 such that:

$$p_3 = [p_{3,x},\ p_{3,y}] = [T_{min}cos(\alpha),\ T_{min}sin(\alpha)] \tag{4.32}$$

Let a, b and c be given as in Fig. 4.7.

$$\alpha_3 = \frac{\pi}{2} - \alpha \tag{4.33}$$

$$c = \frac{p_{3,y}}{tan(\alpha_3)} \tag{4.34}$$

$$b = c + p_{3,x} \tag{4.35}$$

$$a = b * tan(\alpha_3) \tag{4.36}$$

Line l_3 is found to be:

$$y = a - \frac{a}{b}x \tag{4.37}$$

giving the constraint:

$$[-\frac{a}{b},\ -1] \begin{bmatrix} u_x^1 \\ u_y^1 \end{bmatrix} \leq -a \tag{4.38}$$

The same is done for line l_4, and all the constraints for angle and magnitude are stacked.

Note that if u_x or u_y is zero or the point p is in another quadrant, the results are a bit different. Please see the code in App. E to see how this is done.

This was how the rate constraints are found for one actuator at one time step. This has to be done for all the constraints over the time horizon. But the future values of u^k are not known. As an approximation, the last optimal prediction $\mathbb{U}^{0*}$ is used, and the polynomials are constructed around the different values in this prediction.

A special situation occur if $\sqrt{u_x^2 + u_y^2}$ is very small, i.e. close to zero. To prevent jumping when α is zero, this is solved by fix α when $\sqrt{u_x^2 + u_y^2} \leq \epsilon$.

4.4 Comparison with an existing (quasi-)static method

To see how the MPC algorithm perform, it will be compared to an existing allocation method, namely the (quasi-) static method outlined in Sec. 2.2.1. This is a method that is valid for all $\alpha_0 = cte.$, but it is not optimal with respect to a time-varying α. $\boldsymbol{\tau}_{comm} = T(\alpha)\mathbf{f}$. In this case the thrusters are held at fixed positions 90 degrees relative to each other as seen in Fig. 4.4. This results in a constant matrix $T(\alpha) = T(\alpha_0) = B$.

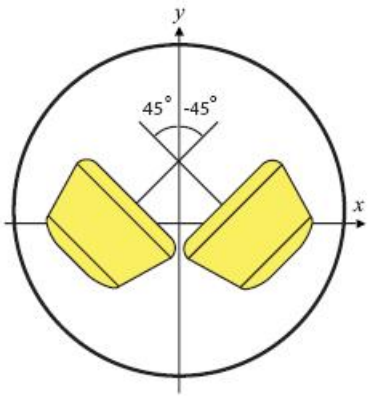


Figure 4.8: Azimuth angles relative to the sylinder(Tyssø and Aga(2006))

The constant actuator conguration matrix is called B to avoid it to be mixed up with the thrust maginitude T.

$$\begin{aligned} J &= \min_{\mathbf{f}} \{\mathbf{f}^T \mathbf{W} \mathbf{f}\} \\ &\quad subject\ to: \boldsymbol{\tau} - \mathbf{B}\mathbf{f} = \mathbf{0} \end{aligned} \tag{4.39}$$

$\mathbf{W}$ is a matrix (positive definite) weighting the control forces. The generalized matrix can then be found as:

$$\mathbf{B}_w^\dagger = \mathbf{W}^{-1}\mathbf{B}^T\left(\mathbf{B}\mathbf{W}^{-1}\mathbf{B}^T\right)^{-1} \tag{4.40}$$

Note that $\mathbf{u}$ is the same as the thrust magnitude T referred to previous this chapter. The weighting matrix $\mathbf{W}$ is chosen to be:

$$\mathbf{W} = \underbrace{\begin{bmatrix} 1 & & 0 \\ & 1 & \\ & \ddots & \\ 0 & & 1 \end{bmatrix}}_{I}, I = 8. \tag{4.41}$$

Then the thrust magnitude $\mathbf{u}$ is given by:

$$\mathbf{u} = \mathbf{K}^{-1}\mathbf{B}_w^\dagger \boldsymbol{\tau}_{comm} \tag{4.42}$$

This allocation method does not enable constraints. To be able to compare it to the MPC algorithm, 'fake' constraints will be added in simulations. That is, if $\mathbf{u}$ is bigger than the last value $\mathbf{u}^0$, the $\mathbf{u}$ applied will be equal to:

$$\mathbf{u}_{app} = \mathbf{u}^0 + \Delta u \tag{4.43}$$

Δu will be the same as the one used in the MPC algorithm (ΔT).

CHAPTER

5

RESULTS

The three MPC-algorithms have been simulated in Matlab. Algorithm 3 has been implemented in C, in order to run it on CyberRig I. Simulations of these algorithms are shown in this chapter.

Note that all angles are calculated with reference to a vessel fixed coordinate system with traditional xy-coordinates. The thrusters mounted on the rig rotate [-180, 180] degrees from the angle they are mounted in. See Fig. 5.1. Angle zero is defined to be at the point [1 0] in the vessel fixed coordinate system. To obtain the same force with thruster 1 135° has to be subtracted. Note that for marine applications is it normal to let x be the longitudinal axis directed from aft to fore, y the transversal axis directed to starboard,

Thruster	*Degrees*	*Radians*
1	-135	$-3\pi/4$
2	-135	$-3\pi/4$
3	+135	$3\pi/4$
4	+135	$3\pi/4$
5	+45	$\pi/4$
6	+45	$\pi/4$
7	-45	$-\pi/4$
8	-45	$-\pi/4$

Table 5.1: Added angles when MPC-algorithm is run on CRI

and x the normal axis directed to bottom.

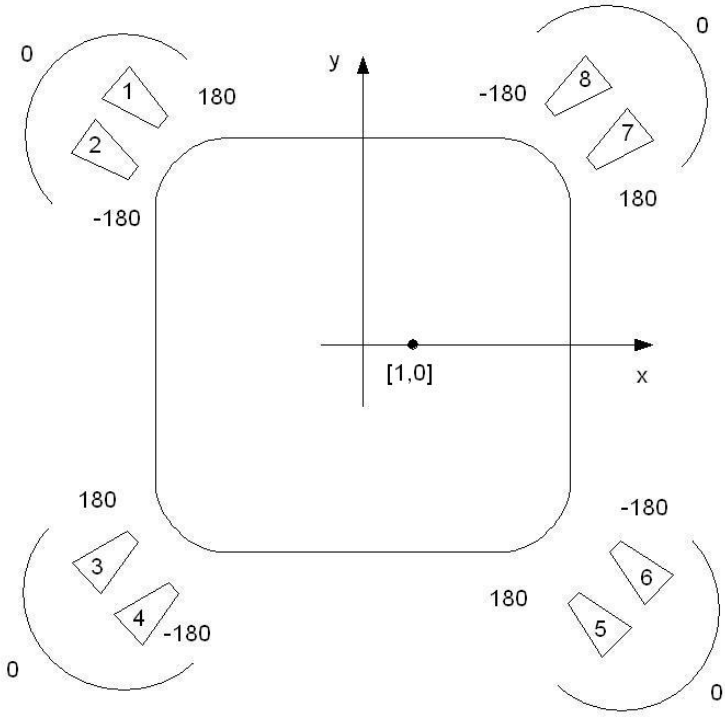


Figure 5.1: Vessel fixed coordinate system and thrusters

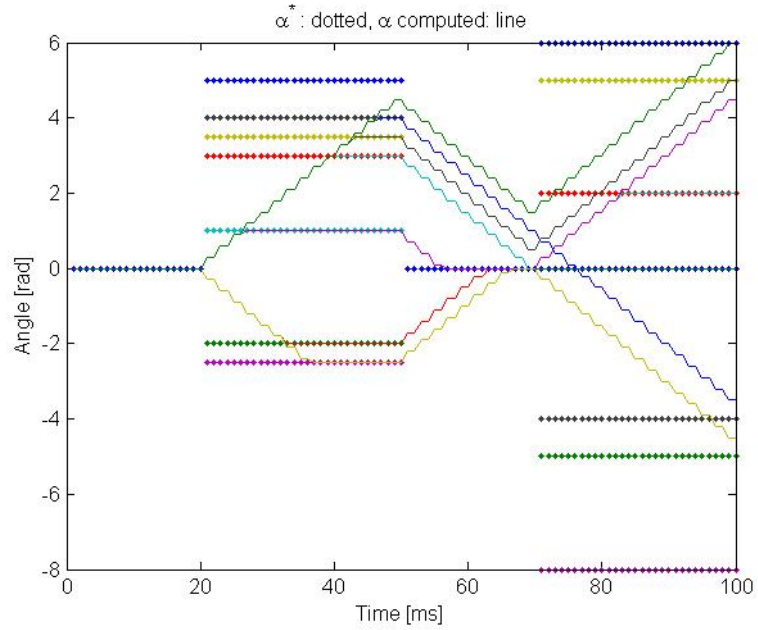
5.1 Algorithm 1

Algorithm 1 was implemented for simulations in Matlab. Fig. 5.2(a) and Fig. 5.2(b) shows the computed α and T where α^* and T^* has been assumed, and there is a step in α^* and T^* for each of the thrusters. There is a maximum of $\Delta\alpha_i = 0.1$ $\Delta T_i = 0.1$ for $i \in [1, N]$ over the whole horizon. Q and R are chosen to be the identity matriz I.

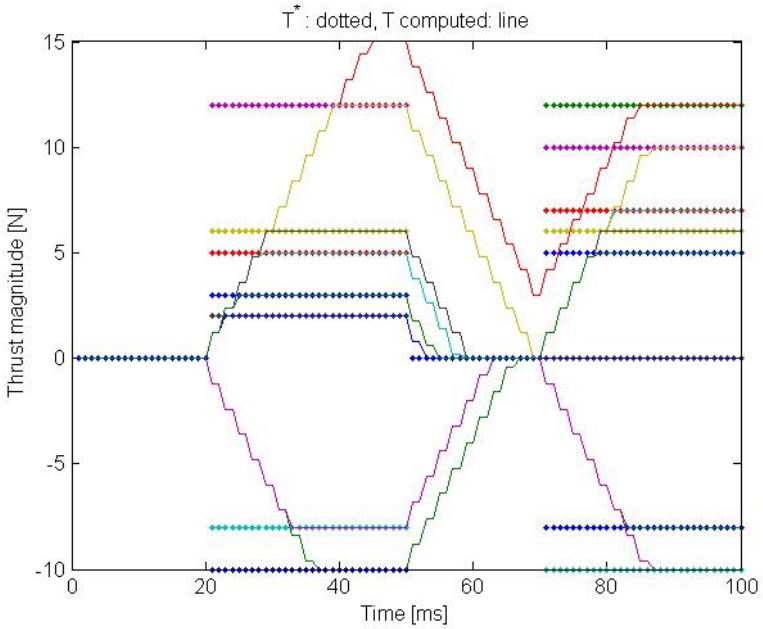
Fig. 5.3(a) and Fig. 5.3(b) shows the computed α and T when the optimal values α^* and T^* have been computed by the algorithm briefly described in Sec. 4.2, and there is a step in the commanded generalized force to $\boldsymbol{\tau}_{comm} = [10, 10, 0]^T$.

With this formulation rate constraints are implemented, which is an improvement from the quasi-static method. Also the thrusters are allowed to rotate. But it is seen that the time horizon N has little effect on the result, and that

in general this formulation does not benefit from the prediction horizon. α and T will go to the 'optimal' values α^* and T^* as fast as the rate constraints let them, and not choose a better path or different goal, knowing the future constraints.

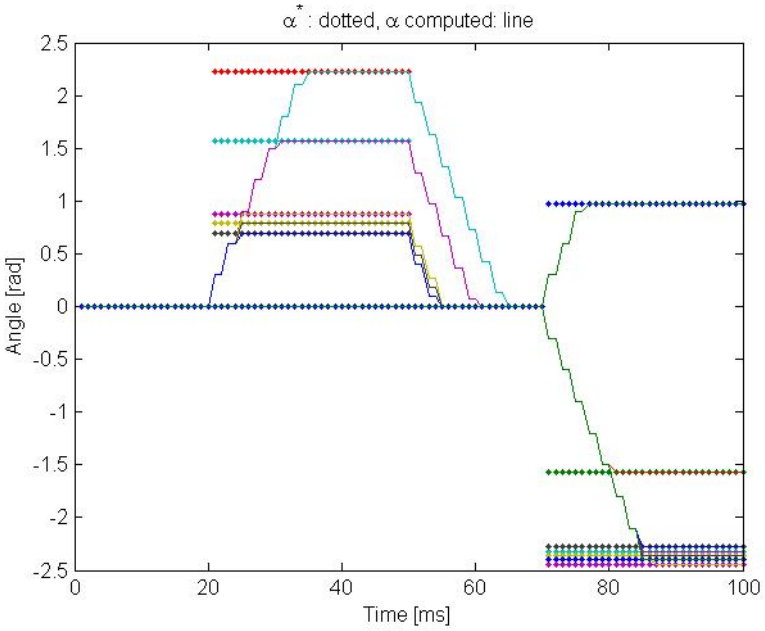


(a) mpc1 - alpha

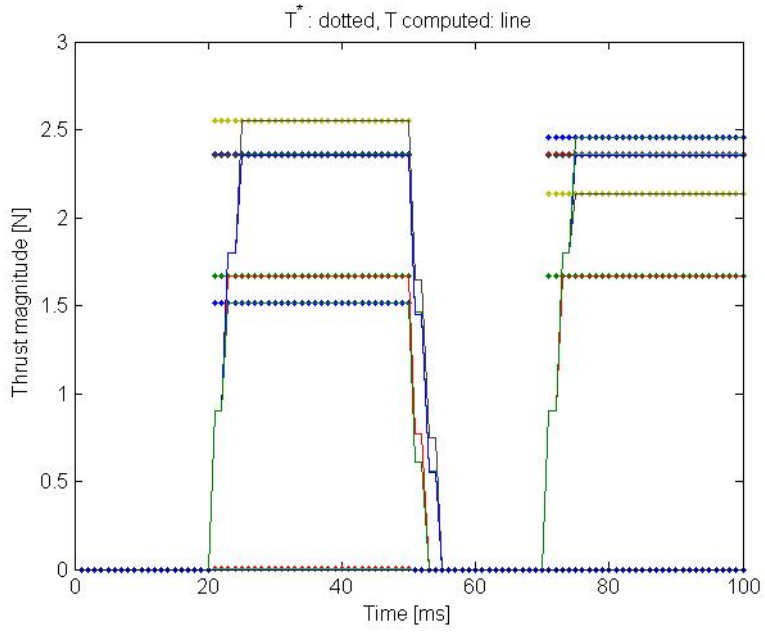


(b) mpc1 - T

Figure 5.2: Algorithm 1, α^* and T^* given



(a) mpc2 - alpha



(b) mpc2 - T

Figure 5.3: Algorithm 1, $\tau_{comm} = [10, 10, 0]^T$

5.2 Algorithm 2

Algorithm 2 was implemented for simulations in Matlab. This algorithm minimizes the error between the commanded force $\boldsymbol{\tau}_{comm}$ and the computed force Bu. The algorithm simulated implements constraints, but in a rather incomplete way. Instead of using the rate constraints from Sec. 4.3.2, they are approximated with a 'moving circle'. This means that the allowed sector to move from one step to another is a circle around $[u_{0,x}u_{0,y}]$ with radius 0.1. Forbidden sectors are also implemented. The simulations are done with a step in $\boldsymbol{\tau}_{comm}$ from $[0,0,0]^T$ to $[10,10,0]^T$, then to $[0,0,0]^T$ again, and lastly to $[-10,-10,0]^T$.

In the results the last weight (diagonal matrix with eight elements) is $Q^N = diag(50)$, while $Q^k = diag(1)$ for $k \in [1, N-1]$. The time horizon is N=6. Fig. 5.4 shows the commanded generalized force, and the resulting force after optimization. Fig. 5.5 and Fig. 5.6 shows the calculated α and T respectively. Fig. 5.7 shows how the actuator 'responds' in an xy-plot. An approximation to the allowed sectors is shadowed.

How variations in the time horizon and the last element of the weighting matrix Q^N affects the results have also been examined. The combinations in the plots are:

-$N = 2$, $Q^N = diag(1)$

-$N = 10$, $Q^N = diag(1)$

-$N = 3$, $Q^N = diag(50)$

-$N = 10$, $Q^N = diag(50)$

A larger time horizon than $N = 10$ is not recommended, as computational efforts increase a lot with increasing time horizont. The relevant figures can be found in App. B. The plots indicates that a short time horizon combined with a high value of Q^N is not a good combination. Apart from that the plots diverges surprisingly little from each other. The reason for this is discovered: The rate constraints are constructed around the last step optimal value $\mathbb{U}^0$. It is assumed that the thrusters initially are positioned to be in the middle their respective allowed sector, with a small thruster magnitude, for all future times in the prediction horizon. This means that $u^{0,*} = u^{1,*} = u^{2,*}$ etc initially. This way the rate constraints become a problem, as the future maximum and minimum values for u^k are all equal to the present max and min values, and the prediction horizon is of little value. The MPC problem is reduced to a normal qp-problem, but with considerably higher computational efforts.

The problem of rate constraints around $\mathbb{U}^0$ and the shortcoming of imple-

menting circles as approximations to the rate constraints are dealt with in the implementation of the next algorithm.

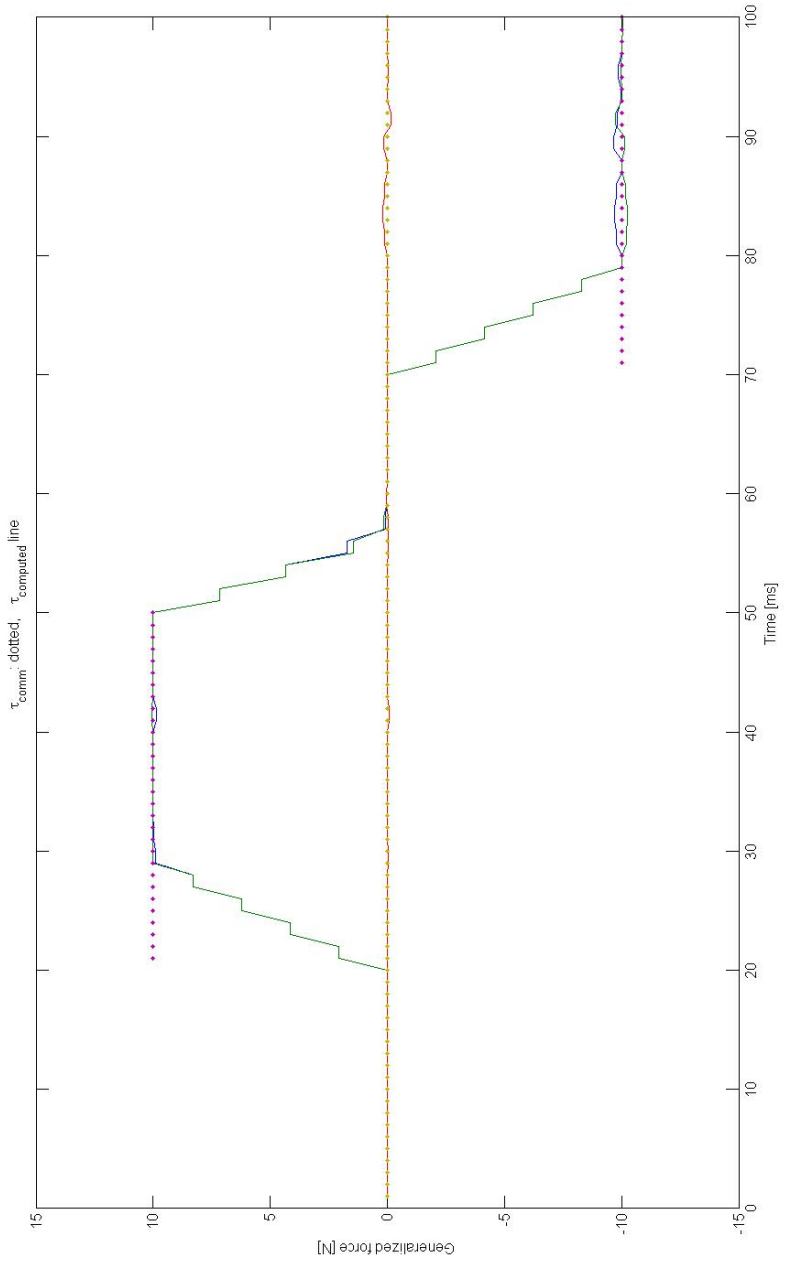


Figure 5.4: Algorithm 2: tau

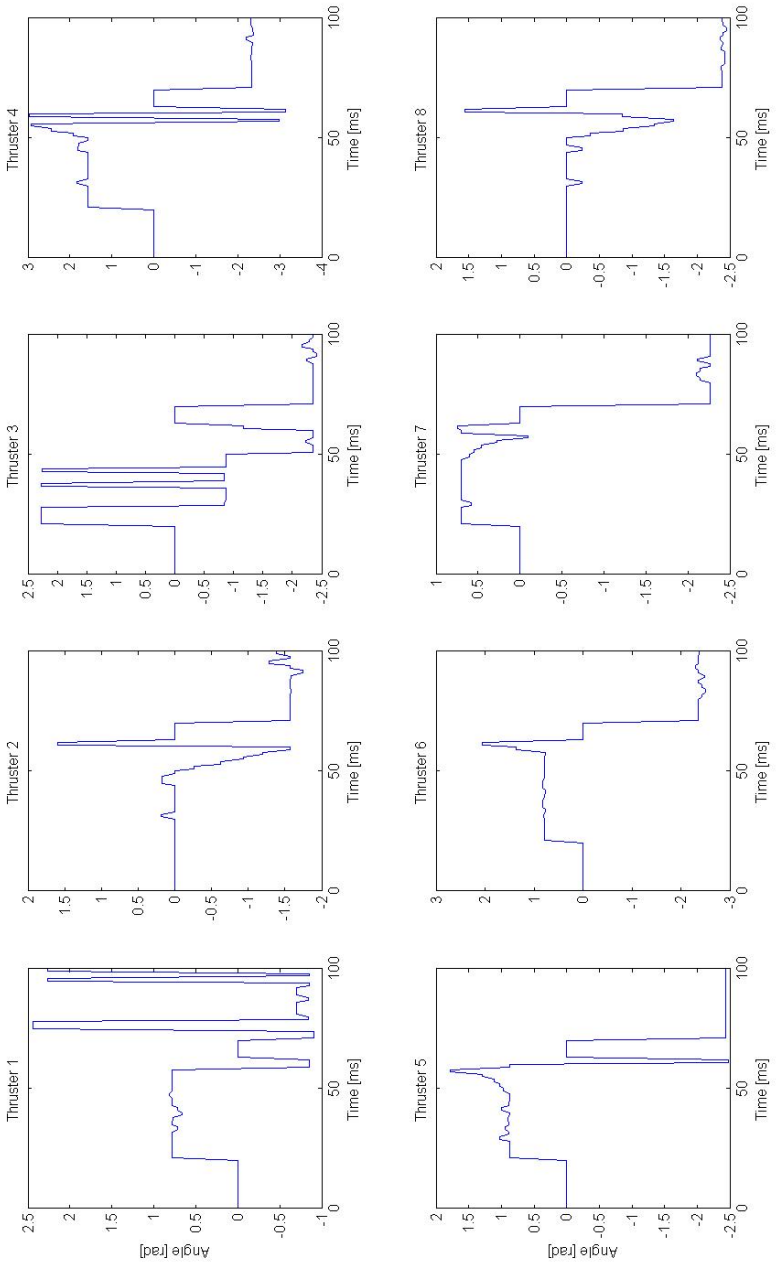


Figure 5.5: Algorithm 2: Alpha

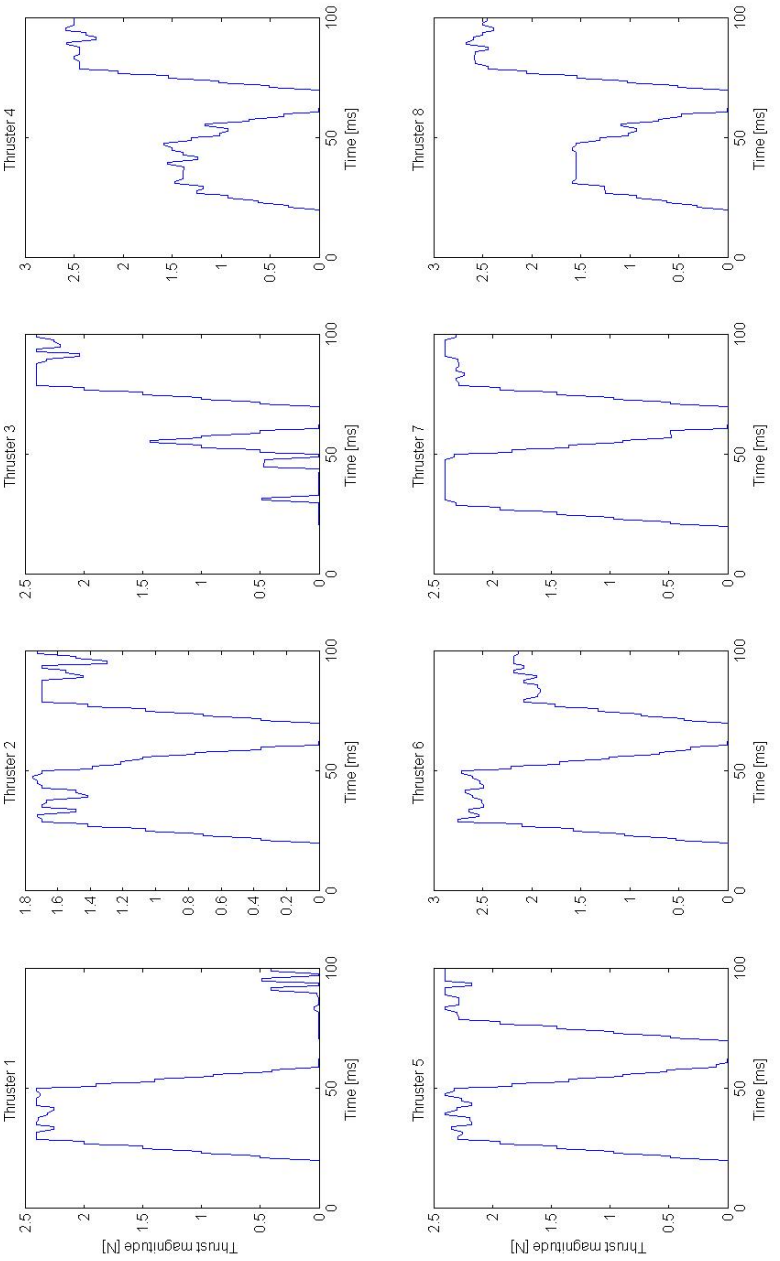


Figure 5.6: Algorithm 2: T

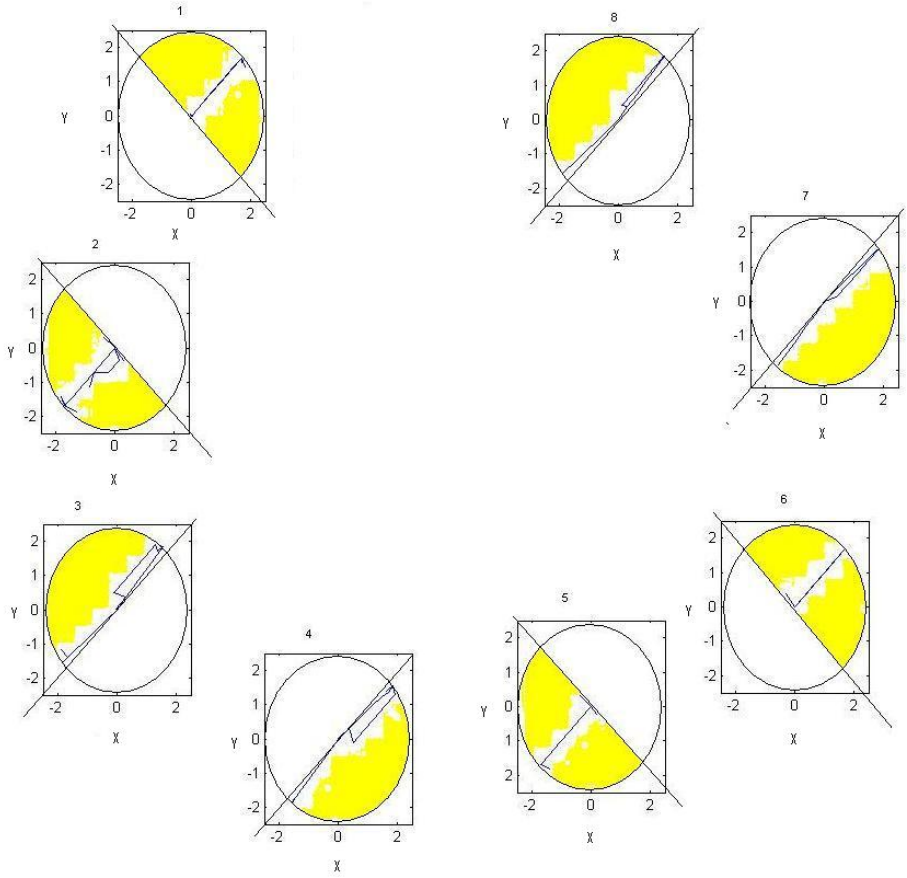


Figure 5.7: Algorithm 2: Allowed sectors for the thrusters

5.3 Algorithm 3

Algorithm 3 is the most interesting. It includes energy minimizing, forbidden sector constraints and the rate constraints from Sec. 4.3.2.

The problem with rate constraints around $\mathbb{U}^0$ is solved with changing the values of $\Delta\alpha$ and ΔT, instead of changing $\mathbb{U}^0$. It is hard to see how $\mathbb{U}^0$ could be predicted initially. $\Delta\alpha$ and ΔT are increased initally, more specific: $\Delta_2\alpha = 2 * \Delta\alpha$ for $k = 2$, $\Delta_3\alpha = 3 * \Delta\alpha$ for $k = 3$ etc.

As algorithm 1 and 2, this algorithm has also been simulated in Matlab. The relevant figures can be found in App. B. This algorithm have been chosen for online implementation. To allow it to run in real-time on the target pc on CyberRig I, the algorithm has been implemented as a S-function in c, with the help of S-function-Builder in Simulink.

The algorithm has been simulated with a time-horizon $N = 3$. The weighting matrices are chosen to be $Q^k = diag(1)$ for $k \in [1, N-1]$, $Q^N = diag(5)$ and $R = diag(25)$. $\Delta\alpha = 0.1$ and $\Delta T = 0.1$. Another good weighting is to increase R^k with increasing k, to ensure that the last element τ^N is the closest possible to $\boldsymbol{\tau}_{comm}$. R should be much bigger than Q anyway, to minimize the slack-variables. In the appendix the simulated τ under a step in the commanded force is also shown (in Fig. B.0.2).

Fig. 5.8 shows tau commanded and tau mpc, where $\boldsymbol{\tau}_{comm}$ is generated by a joystick. The sampling time is 0.1 s.

Fig. 5.10 visualizes how the forces are kept within the allowed sectors, while Fig. 5.9 shows a detail of T and alpha. It is noticed that the constraints are overheld, at the same time as the referance is followed.

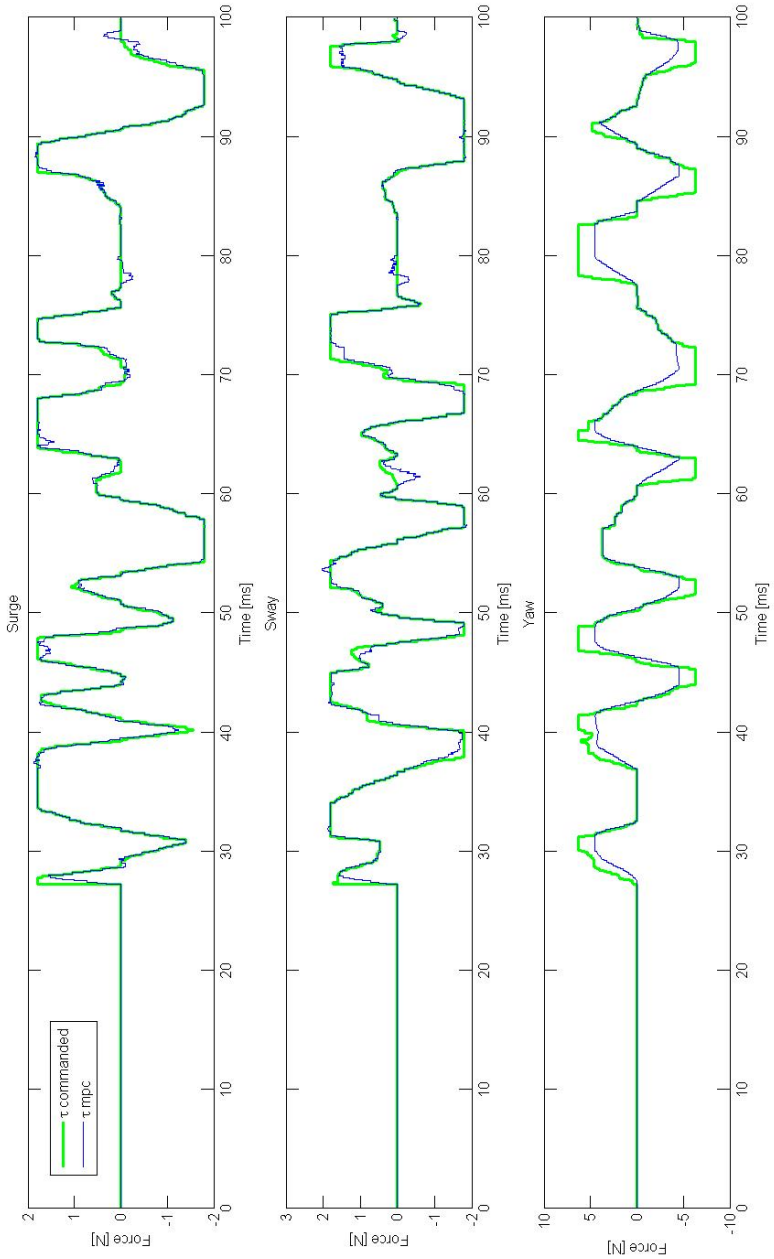


Figure 5.8: Tau commanded and tau mpc

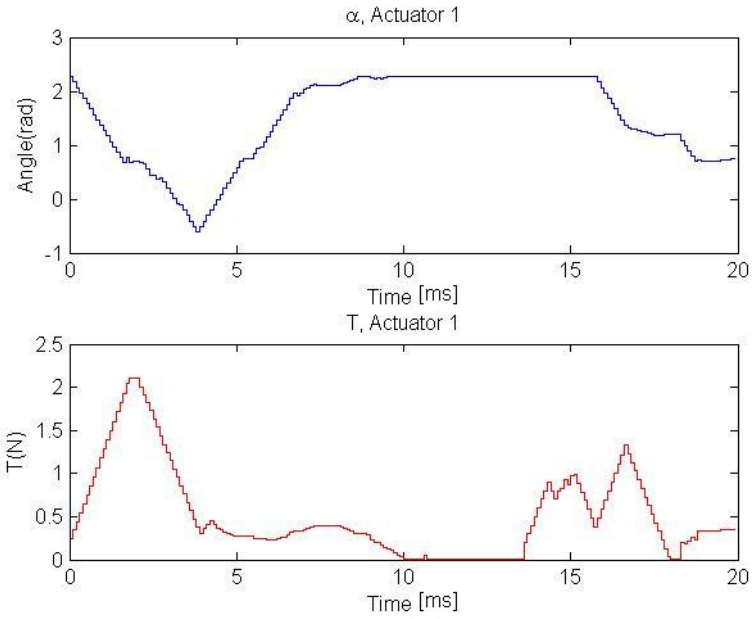
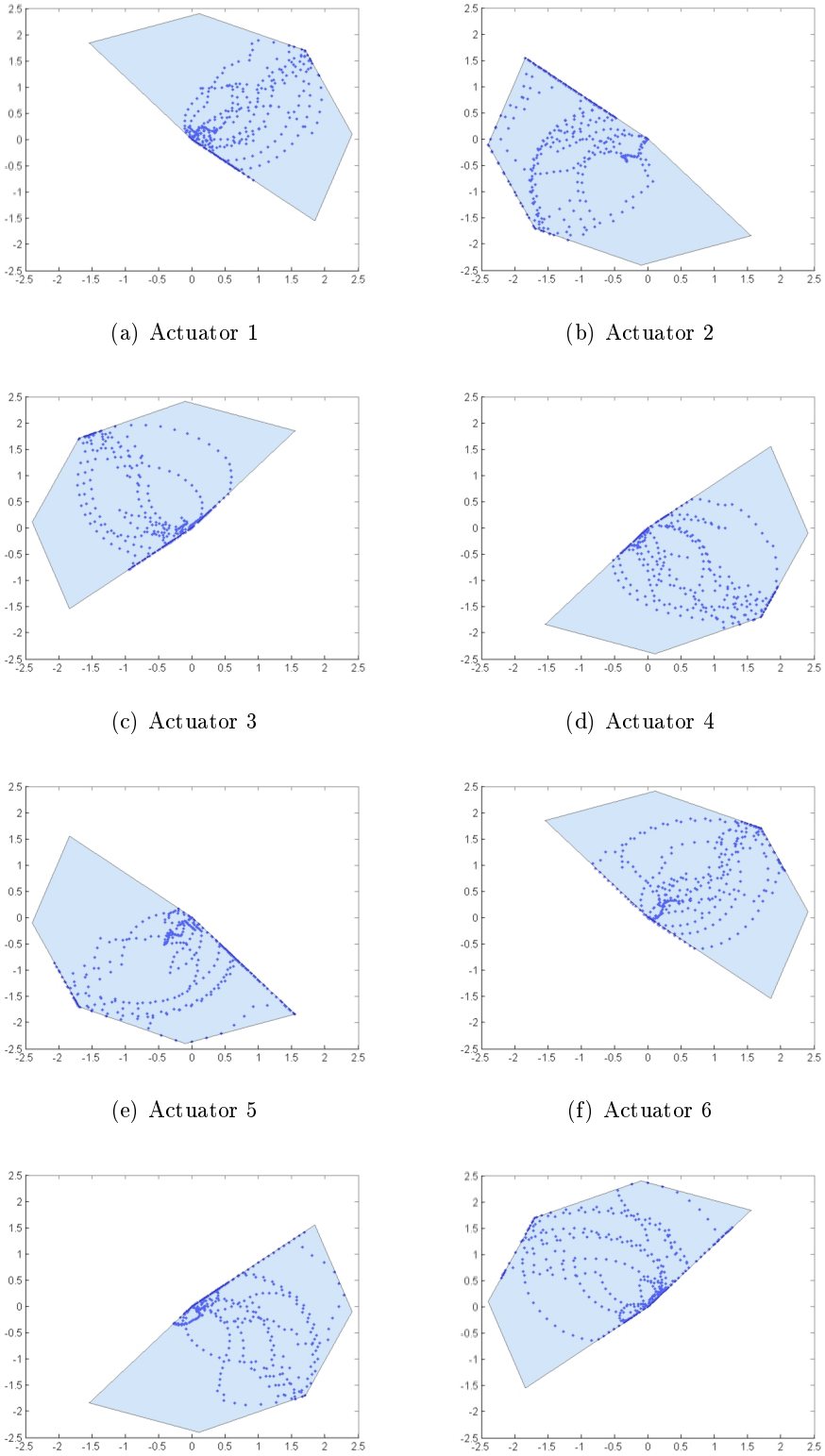


Figure 5.9: Detail showing T and alpha



(a) Actuator 1 (b) Actuator 2

(c) Actuator 3 (d) Actuator 4

(e) Actuator 5 (f) Actuator 6

(g) Actuator 7 (h) Actuator 8

Figure 5.10: Simulated force for each of the actuators and the corresponding allowed sectors using mpc control allocation

5.4 MPC-CA vs. quasi-static-CA

Fig. 5.12 shows the simulated force for each of the actuators using the quasi-static control allocation described in Sec. 4.4. Clearly by using this method it is no way to implement forbidden sectors. One has to calculate optimal values for T, and later on limit them. This way one can not assure that the computed force vector τ is correct, moreover, it will probably not be correct. For example, in order to avoid that stream from one thruster is pumped into the other at the same leg and result in nonlinear interactions, T^* should be greater then or equal to zero.

Fig. 5.11 shows the resulting force when quasi-static CA is used with T^* saturated between [0, 2.4] (which is the limit for T^* also in the MPC CA). Obviously the resulting force is only the half of the commanded force, as negative T is not allowed.

Fig. 5.13 shows the commanded force, the force computed by the MPC-algorithm and the force using the quasi-static formulation. Tau commanded is generated by a joystick. For the quasi-static algorithm 'fake' rate constraints as descibed in Sec. 4.4 are applied, but there are no limit on the minimum and maximum values for α and T.

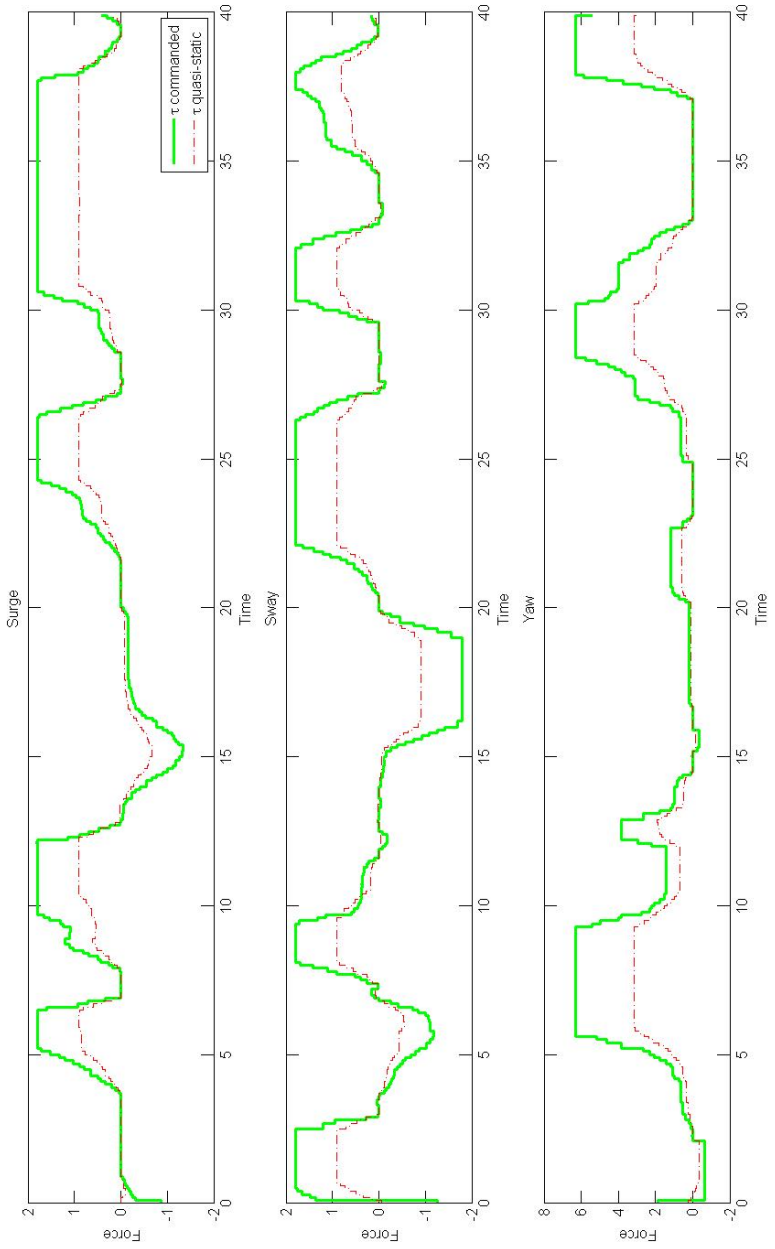
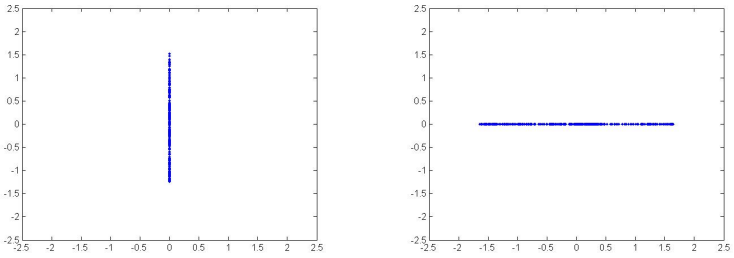
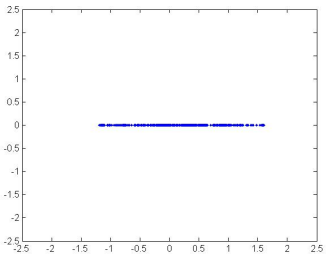


Figure 5.11: Simulated force by using quasi-static control allocation with saturation

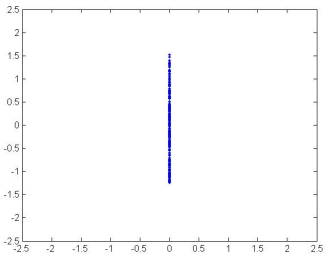


(a) Actuator 1

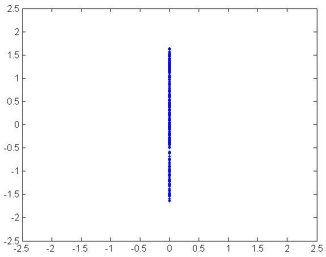
(b) Actuator 2



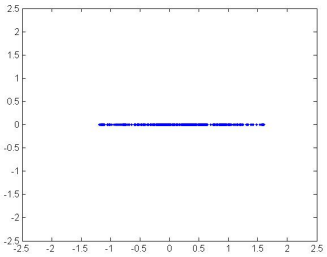
(c) Actuator 3



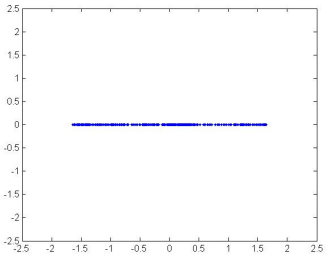
(d) Actuator 4



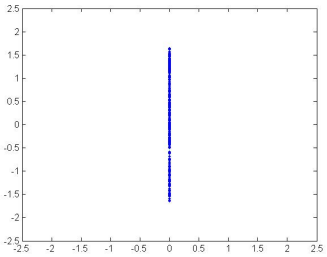
(e) Actuator 5



(f) Actuator 6



(g) Actuator 7



(h) Actuator 8

Figure 5.12: Simulated force for each of the actuators using quasi-static control allocation

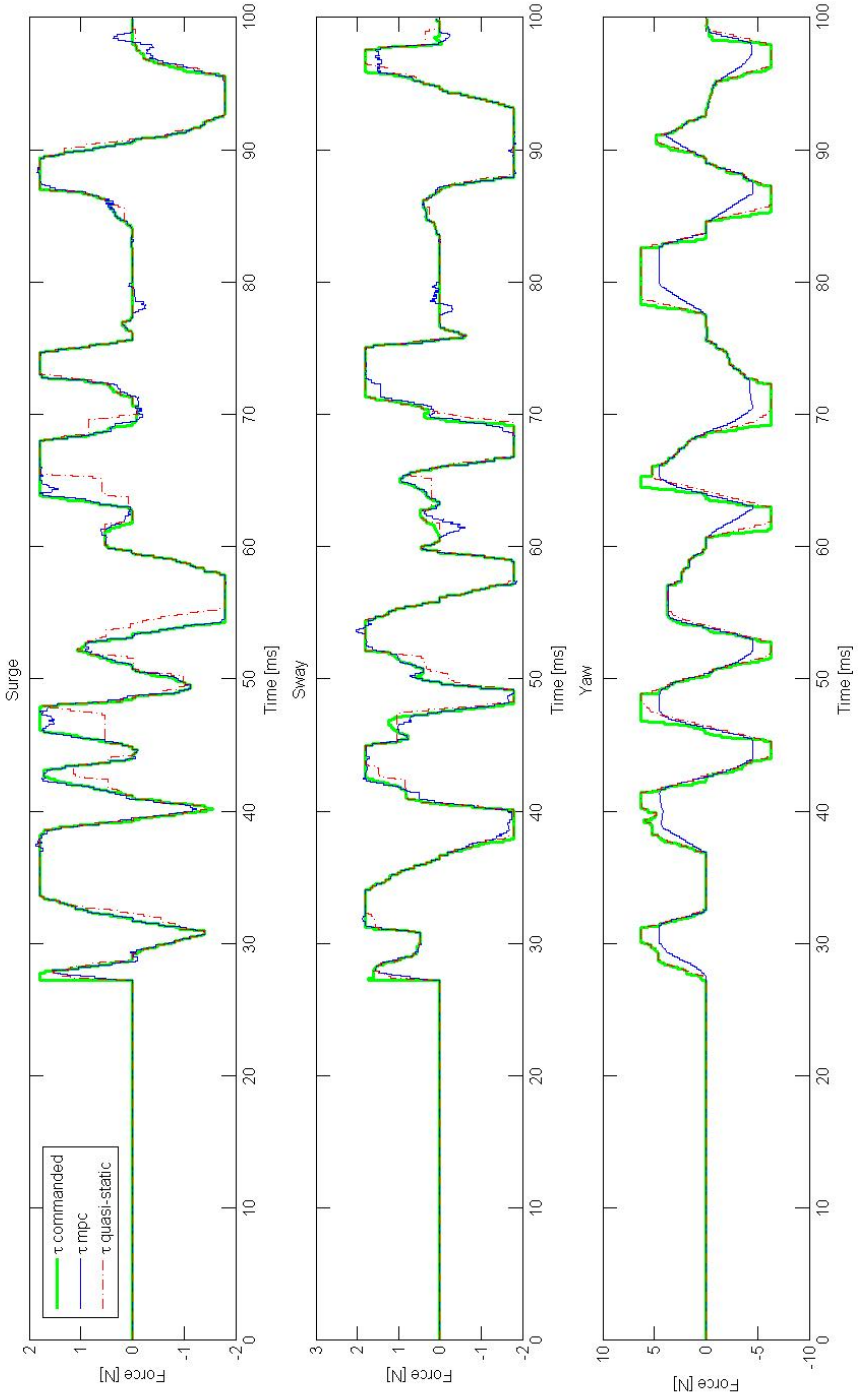


Figure 5.13: Tau commanded, tau mpc and tau quasi-static

CHAPTER

6

DISCUSSION OF RESULTS

In the previous chapters 3 MPC algorithms for control allocation have been suggested and simulated. Of them, algorithm 3 is considered the best and most complete. It was the algorithm elected for online implementation. Therefore Algorithm 3 forms the basis for the analysis in this chapter. In this chapter advantages and disadvantages are beeing discussed, and a comparision between MPC control allocation and quasi-static control allocation is carried out.

As seen in Sec. 5.3 the MPC algorithm performs very well and follows the reference given by $\boldsymbol{\tau}_{comm}$. With a step in $\boldsymbol{\tau}_{comm}$ the applied thruster magnitude and angle results in an slowly rising computed τ, as the rate constraints must be overheld. It is also seen how the MPC algorithm keeps the force magnitude and angle for each azimuth thruster within the allowed sector.

As seen in the figures in Sec. 5.4 both the MPC algorithm and the quasi-static algorithm follow the reference, the commanded generalized force $\boldsymbol{\tau}_{comm}$ quite well. Sometimes MPC performs best, and at other times the quasi-staic is better. It seems that the quasi-static method gives smoother results with less oscillations. This can come of inaccuracies in the implementation in c, as some things have had to be solved ad hoc as problems came along.

However, there are various benefits of using MPC over the quasi-static method in Control Allocation. Most important is the ability of MPC to handle constraints. Rate constraints and forbidden sectors have been implemented with success. Another benefit is that MCP allows for fault tolerant allocation, al-

though this has not been done in this thesis. If one of the thrusters fail, its associated control $[u_x, u_y]$ can be set to zero in the equality constraints, and the other constraints will take care of obtaining the commanded force. Another option is to remove all the associated controls from the problem formulation.

If the future reference signal is known, this can be incorporated in MPC, and resulting in a better response. In the implementation the future reference was assumed to be equal to the present, but this is easily changed. Using the quasi-static method it is not possible to include the known future reference.

However, the MPC method requires a considerably greater computational effort. The quasi-static method is very easy to implement, and does not require much storage space or computational efforts. Allowing for the thusters to rotate will probably increase mechanical wear and tear. But the power consumption is minimized in the cost function, and rotation may lead to that $\boldsymbol{\tau}_{comm}$ is reached faster, and gives greater flexibility.

So, what are the main drawbacks of using MPC in control allocation? First of all the computational efforts are big. Large matrices requires storage space, and demanding computations have to be carried out online. Updating the matrices is a time-consuming task, and increased time horizon for the prediction increases the time required for qp-calculation drastically. It it easy to imagine that an increase in the prediction horizon will result in a qp-problem that cannot be solved in real-time.

Secondly the rate constraints are constructed around an unknown future state for the actuators. In the algorithm, the future values were assumed to be $\mathbb{U}^{*0}$, or the last optimal trajectory computed. This is problematic for two reasons. 1) It is not known whether the approximation $\mathbb{U}^{*0}$ is correct. But control allocation is a open-loop problem and the thruster dynamics is included in the formulations (as rate constraints), so the difference between what is applied and what the result is should not be so big. 2) There is the problem of initiation of $\mathbb{U}^{*0}$ and rate constraints. In the implementation it is assumed that the thrusters initially are positioned to be in the middle their respective allowed sector, with a small thruster magnitude, for all future times in the prediction horizon. Instead of changing the $\mathbb{U}^{*0}$ that the constraints are made around, the rate constraints are made bigger linearly with increasing k. This is not optimal, as larger jumps in α and T than desired are made possible.

CHAPTER

7

CONCLUSION AND FURTHER WORK

7.1 Conclusion

Three Model Predictive Control algorithms for control allocation have been proposed. Of them, Algorithm 3 is considered to be the most complete. It minimizes energy, and implements constraints for forbidden sectors and rate constraints. Simulations shows that the commanded generalized force is followed with this method.

In comparison with a quasi-static method the MPC algorithm perform very well. Forbidden sectors and rate constraints are implemented with MPC, and this is not possible with the quasi-static method. The quadric programming approach that MPC utilize also yields a lower energy consumption compared to using generalized inverces with fixed angles. The main drawbacks using MPC in control allocation are complexity and larger on-line computational efforts.

7.2 Further work

There are several possible threads for futher research on the topic:

- A formulation based on 3 DOF has been developed. An extencion to 6 DOF could be carried out.
- The MPC algorithm has been compared with a quasi-static method for control allocation. It would be interessting to see how MPC perform in comparison other more complex constrained methods, such as linear quadratic formulation from Sec. 2.2.2 or a nonlinear method allowing for azimuthing thrusters. Especially benefits of mpc (if any) vs increased computational effort is interesting.
- The present cost function penalizes change in thrust magnitude and in azimuth angle, to reduce the power consuption. A more complete cost function can be formulated.
- The problem with initializing $\mathbb{U}^{*0}$ /create consistent rate constraints over time horizon should be addressed.

REFERENCES

Allgöver, F., Findeisen, R., Zolatan, K.N. (2004), *Nonlinear Model Predictive Control: From Theory to Application*, J. Chin. Ins. Chem. Engrs, Vol 35, No.3, pp. 299-315.

Breivik, M. (2007), *Introductory Lecture in TTK4190 Guidance and Control*, Lecture Notes, TTK4190 Guidance and Control - Spring 2007, NTNU

Brown, D.T. and Ekstrom, l. (2005), *Vessel thruster-thruster interactions during azimuthing operations*, 24th International Conference on Offshore Mechanics and Arctic Engineering (OMAE).

Fossen, T.I. (2002), *Marine Control sytems*, Marine Cybernetics AS, P.O.Box 4607, 7451 Trondheim, Norway.

Fossen, T.I. and Johansen T.A. (2006), *A survey of Control Allocation Methods for Ships and Underwater vehicles*, 14th IEEE Mediterranean Conference on Control and Automation. Acona, Italy.

Johansen, T.A, Fuglseth, T.P, Tøndel, P. and Fossen, T.I (2003), *Optimal constrained control allocation in marine surface vessels with rudders*, IFAC Conf. Manoeuvring and Control of Marine Craft, Girona.

Johansen, T.A, Fossen, T.I, Berge, S.P (2004b), *Constraint Nonlinear Control Allocation with Singularity Avoidance using Sequential Quadratic Programming*, IEEE Transactions on Contol Systems Technology, Vol.TCST-12, pp. 211-216.

Johansen, T.A. (2004), *Optimizing Nonlinear Control Allocation*, IEEE Conf.

Decision and Control, Nassau, Bahamas, pp. 3435-3440.

Johansen, T.A, Fossen, T.I. and Tøndel, P (2005), *Efficient Optimal Constrained Control Allocation via Multi-Parametric Programming*, AIAA Journal of Guidance, Control and Dynamics, Vol. 28, pp. 506-515.

Kalman, R.E. (1960a), *Contributions to the theory of optimal control*, Boletin de la Societe Methematique de Mexicana, Vol 5, pp.102-119.

Kalman, R.E. (1960b), *A new approach to linear filtering and prediction problems*, Transactions of ASME, Journal of Basic Engineering, Vol 82, Series D, pp. 35-45.

Maciejowski, J.M (2002), *Predictive Control with Constraints*, Prentice Hall

Mayne, D.Q, Rawlings, J.B, Rao, C.V, Scokaert, P.O.M (2000), *Constrained model predictive control: Stability and optimality*, Automatica, Vol. 36, pp. 789-814.

Plumlee, J.H and Bevly, D.M. (2004) *Control of a Ground Vehicle using Quadratic Programming Based Control Allocation Techniques*, Proceeding of the American Control Conference, Boston, Massachusetts June 30-July 2.

Qin, S.J. and Badgewell, T.A. (2003), *A survey of industrial model predictive control technology*, Control Engineering Practice, Vol. 11, pp. 733764.

Sørdalen, O.J. (1997), *Optrimal thrust allocation for marine vessels*, Control Engineering Practice, Vol. 5, pp. 1223-1231.

Spjøtvold, J. and Johansen, T.A. (2007), Constrained Control Allocation for a Semi-Submersible Drilling Unit via Parametric Programming

Torpe, H. (2007), *Pipeline Liquid Control using Nonlinear MPC and OLGA*, Master's thesis, Norwegian University of Science and Technology (NTNU).

Tyssø, J. and Aga, A.H. (2006), *DP Control System Design for CyberRig I*, Master's thesis, Norwegian University of Science and Technology (NTNU).

Tøndel, P., Johansen, T.A. and Bemporad, A. (2003), *An algorithm for multi-parametric quadratic programming and explicit MPC solutions*, Automatica, Vol. 39, pp. 489-497.

APPENDIX

A

CODE

Algorithm 3: mpc_varying_wrapper.c

Wrapper-file in c. for Algorithm 3. The fixed constraints (equality constraints and forbidden sectors) included in matrices in header-file matrices2.h.

```
/*
  *
  *   --- THIS FILE GENERATED BY S-FUNCTION BUILDER: 3.0 ---
  *
  *   This file is a wrapper S-function produced by the S-Function
  *   Builder which only recognizes certain fields.  Changes made
  *   outside these fields will be lost the next time the block is
  *   used to load, edit, and resave this file. This file will be overwritten
  *   by the S-function Builder block. If you want to edit this file by hand,
  *   you must change it only in the area defined as:
  *
  *        %%%-SFUNWIZ_wrapper_XXXXX_Changes_BEGIN
  *            Your Changes go here
  *        %%%-SFUNWIZ_wrapper_XXXXXX_Changes_END
  *
  *   For better compatibility with the Real-Time Workshop, the
  *   "wrapper" S-function technique is used.  This is discussed
  *   in the Real-Time Workshop User's Manual in the Chapter titled,
```

```
  *   "Wrapper S-functions".
  *
  *   Created: Thu Nov 15 19:33:50 2007
  */


/*
 * Include Files
 *
 */
#if defined(MATLAB_MEX_FILE)
#include "tmwtypes.h"
#include "simstruc_types.h"
#else
#include "rtwtypes.h"
#endif
/* %%%-SFUNWIZ_wrapper_includes_Changes_BEGIN --- EDIT HERE TO _END */
#include <math.h>
#include "matrices2.h"
#include "constrAT.h"

void c2f(double *in, double *out, int n, int m)
{
    int i, j;
    for(i=0;i<n;i++){
        for(j=0;j<m;j++){
            out[j*n+i] = -in[i*m+j];
        }
    }
}
/* %%%-SFUNWIZ_wrapper_includes_Changes_END --- EDIT HERE TO _BEGIN */
#define u_width 3
#define y_width 1
/*
 * Create external references here.
 *
 */
/* %%%-SFUNWIZ_wrapper_externs_Changes_BEGIN --- EDIT HERE TO _END */
/* extern double func(double a); */
/* %%%-SFUNWIZ_wrapper_externs_Changes_END --- EDIT HERE TO _BEGIN */

/*
 * Output functions
 *
```

```
 */
void mpc_varying_Outputs_wrapper(const real_T *tau,
                          real_T *y0,
                          real_T *y1,
                          real_T *y2,
                          const real_T  *xD)
{
/* %%%-SFUNWIZ_wrapper_Outputs_Changes_BEGIN --- EDIT HERE TO _END */


#define ME 9  //:     NUMBER OF EQUALITY CONSTRAINTS.
#define N 57   //:    NUMBER OF VARIABLES.
#define M 249  //:    TOTAL NUMBER OF CONSTRAINTS
#define MMAX M+1//:   ROW DIMENSION OF A. MMAX MUST BE AT LEAST ONE AND GREATER  THAN M.
#define NMAX N+1//:   ROW DIMENSION OF C. NMAX MUST BE GREATER OR EQUAL TO N.
#define MNN M+N+N// : MUST BE EQUAL TO M + N + N.
#define LWAR 6*(3*NMAX*NMAX/2 + 10*NMAX + 2*MMAX)+30000
#define LIWAR 5000

double xxx[N], lambda[MNN], war[LWAR];
double d[NMAX], xl[N], xu[N];
int m = M; // Number of constraints
int me = ME;     // Number of equality constraints
int mmax = MMAX; // Max number of constraints
int n = N; // Nuber of variabels
int nmax = NMAX; // Max nuber of varabels
int mnn = MNN;
int iout = 1;
int iprint = 1; // No error messages
int lwar = LWAR; // Dimension of real working array
int liwar = LIWAR; // Dimension of integer working array
int ifail, iwar[LIWAR];
double eps1 = 1.0e-15; // Machine precision

int jj, ii, kk, nn, mm, i;
double T, alpha, ux, uy, temp, temp2;
// FILE *fp, *fp2, *fp3, *fp4;
 char linje[20];   /* string with space for 20 signs */
double Auneq3[97][58];//
double buneq3[97];
double Atotal[250][58];
double Atotal_f[58][250];
double btotal[250];
double Any[58][154];
```

```
//to update Auneq3:
double G[4][2], g[4];
double k1,k2,T_temp,alpha_temp, delta, k1_forrige, k2_forrige;
int actuator, index, ind1, ind2, index0;
int na=8;
int HZ=3;
i=xD[66];

//tau commanded
b[0]=tau[0];
b[1]=tau[1];
b[2]=tau[2];
b[3]=tau[0];
b[4]=tau[1];
b[5]=tau[2];
b[6]=tau[0];
b[7]=tau[1];
b[8]=tau[2];


//lower and upper bounds
//(but already taken care of in allowed sector constraints):
for(ii=0;ii<n;ii++){
  xl[ii]=-10;
  xu[ii]=10;
}

////Auneq3: rate constraints:

  for(ii=0;ii<97;ii++){
      for(jj=0;jj<58;jj++){
          Auneq3[ii][jj]=0.0;
     }
      buneq3[ii]=0.0;
  }
  k1_forrige=0.0;
  k2_forrige=0.0;
  delta=0.1;
  for(kk=0;kk<HZ;kk++){
   for(jj=0;jj<na;jj++){ //j is the number of the present actuator
       index=kk*2*na+jj*2;
       k1=xD[index];
       k2=xD[index+1];
```

```
if (fabs(k1)>0.01 || fabs(k2)>0.01){
    actuator=0;
    delta=0.1;
    constrAlphaT(&G[0][0], &g[0], k1, k2, actuator, delta);
}else if (fabs(k1)<=0.1 && fabs(k2)<=0.1){      //if close to zero
if(i==1){//first round, have to initalize
         delta=0.1;
        actuator=jj+1;
        constrAlphaT(&G[0][0], &g[0], k1, k2, actuator, delta);
}else if(i!=1){   //let constraint be equal to last constraint
        alpha_temp=xD[57+jj];     //keep the angle from the last sample

        if(kk==1){
          T_temp=0.205;  //small T, to construct the constraint
          delta=0.2;
        }
        else if(kk=2){
          T_temp=0.305;
          delta=0.3;
        }else{
          T_temp=0.1;
          delta=0.1;
        }

        k1=T_temp*cos(alpha_temp);
        k2=T_temp*sin(alpha_temp);
        actuator=0;

        constrAlphaT(&G[0][0], &g[0], k1, k2, actuator, delta);
        delta=0.1;
}
}

//put in constraints for each actuator
ind1=kk*4*na+jj*4;
ind2=kk*2*na+jj*2;
    for(nn=0;nn<4;nn++){
        for(mm=0;mm<2;mm++){
        Auneq3[ind1+nn][ind2+mm]=G[nn][mm];
        }
        buneq3[ind1+nn]=g[nn];
    }
```

```
   }//end for jj
  }//end for kk
 //end Auneq3



//making the whole matrices Atotal and btotal with all the constraints:
//A: equality constraints and allowed sector constraints
     for(ii=0;ii<153;ii++){
        for(jj=0;jj<57;jj++){
          Atotal[ii][jj]=A[ii][jj];
        }
     }
      for(ii=153;ii<249;ii++){
        for(jj=0;jj<57;jj++){
          //Atotal[ii][jj]=Auneq3_u[ii-153][jj];
          Atotal[ii][jj]=Auneq3[ii-153][jj];
        }
     }

     for(jj=0;jj<153;jj++){
          btotal[jj]=b[jj];
        }
     for(jj=153;jj<249;jj++){
          //btotal[jj]=buneq3_u[jj-153];
          btotal[jj]=buneq3[jj-153];
        }




c2f(&Atotal[0][0], &Atotal_f[0][0], mmax, nmax); //c to fortran (for ql0001_)



//////////////////////end constraints//////////////////////////



//beq=kron(ones(N,1),tau);
//f=[u_horizon zeros(1,3*N)]*H2;  %zeros because of slackvariables
 for(ii=0;ii<N;ii++){
         for(jj=0;jj<N;jj++){
             temp=temp+H2[ii][jj]*xD[jj];
```

```
        }
        d[ii]=temp;
        temp=0;
 }
 iwar[0] = 0;
 ql0001_(&m,&me,&mmax,&n,&nmax,&mnn,&H[0][0],&d[0],&Atotal_f[0][0],&btotal[0],...
  ...&xl[0],&xu[0],&xxx[0],&lambda[0],&iout,&ifail,&iprint,&war[0],&lwar,...
  ...&iwar[0],&liwar,&eps1);


//The solution is the first 16 values in xxx. (ux1, uy1, ux2, uy2...ux16, uy16)
ii=0;
kk=0;
jj=0;
while(ii<16){
    jj=ii+1;

    ux=xxx[ii];
    uy=xxx[jj];
    temp2=ux*ux+uy*uy;
    T=sqrt(temp2);
      if( temp2<0.01 ){   //keep angle if close to zero
          alpha=xD[57+kk];
      }else{
          alpha=atan2(uy,ux);
      }
    temp2=ux*ux+uy*uy;
    T=sqrt(temp2);
    y0[kk]=alpha;       //y0 contains alpha(8 of each)
    y1[kk]=T;           //y2 contains T
    ii=ii+2;
    kk=kk+1;
}

for(ii=0;ii<48;ii++){
     y2[ii]=xxx[ii];     //u over time horizon
}


/* %%%-SFUNWIZ_wrapper_Outputs_Changes_END --- EDIT HERE TO _BEGIN */
}

/*
  * Updates function
```

```
  *
  */
void mpc_varying_Update_wrapper(const real_T *tau,
                          const real_T *y0,
                          const real_T *y1,
                          const real_T *y2,
                          real_T *xD)
{
  /* %%%-SFUNWIZ_wrapper_Update_Changes_BEGIN --- EDIT HERE TO _END */
 int ii;

              for(ii=0;ii<57;ii++){
                   if(ii<48){
                       xD[ii]=y2[ii];   //u
                   }
                   else{
                       xD[ii]=0; //s
                   }
               }


              for(ii=57;ii<65;ii++){
                   xD[ii]=y0[ii-57];    //alpha
               }

  xD[66]=xD[66]+1;
/* %%%-SFUNWIZ_wrapper_Update_Changes_END --- EDIT HERE TO _BEGIN */
}
```

Algorithm 3: constrAT.h

```
void constrAlphaT(double *A, double *but, double u01, double u02, ...
                                    ... int actnr, double del){

double u1,u2;
double poly3[3],poly4[3];
double dL,dalpha,angle,L,l3,l4;
double a,b,c,alpha3,alpha4;
double p3[2],p4[2];
double anglem, anglep;
double point1[2], point2[2];
```

```
double poly[4][3];


double pi=3.1416;

if(actnr==0){
u1=u01;
u2=u02;
}
else if(actnr==1 || actnr==2){
u1=-0.1;
u2=0.1;
}
else if(actnr==3 || actnr==4){
u1=-0.1;
u2=-0.1;
}
else if(actnr==5 || actnr==6){
u1=0.1;
u2=-0.1;
}
else if(actnr==7 || actnr==8){
u1=0.1;
u2=0.1;
}

// Legal change in angle and thrust magnitude
dL=del;
dalpha=del;

angle=atan2(u2,u1);
L=sqrt(u1*u1+u2*u2);

l3=L-dL;
l4=L+dL;


if (u1==0){
    if (u2>0){
        //poly3=[0 -1 -l3];
        //poly4=[0 1 l4];
        poly3[0]=0;
        poly3[1]=-1;
        poly3[2]=-l3;
```

```
        poly4[0]=0;
        poly4[1]=1;
        poly4[2]=l4;
}
    else if (u2<0){
        //poly3=[0 1 -l3];
       // poly4=[0 -1 l4];
        poly3[0]=0;
        poly3[1]=1;
        poly3[2]=-l3;
        poly4[0]=0;
        poly4[1]=-1;
        poly4[2]=l4;
}
}
else if (u2==0){
    if (u1>0){
        //poly3=[-1 0 -l3];
        //poly4=[1 0 l4];
         poly3[0]=-1;
        poly3[1]=0;
        poly3[2]=-l3;
        poly4[0]=1;
        poly4[1]=0;
        poly4[2]=l4;
}
    else if (u1<0){
        //poly3=[1 0 -l3];
       // poly4=[-1 0 l4];
        poly3[0]=1;
        poly3[1]=0;
        poly3[2]=-l3;
        poly4[0]=-1;
        poly4[1]=0;
        poly4[2]=l4;
}
}

else{
    if (l3>0){
        p3[0]=l3*cos(angle);
        p3[1]=l3*sin(angle);
        alpha3=(pi/2)-angle;
        c=p3[1]/tan(alpha3);
```

```
        b=c+p3[0];
        a=b*tan(alpha3);
        //poly3=[-a/b -1 -a];
        poly3[0]=-a/b;
        poly3[1]=-1;
        poly3[2]=-a;

        if (u2<0){
            poly3[0]=-poly3[0];
            poly3[1]=-poly3[1];
            poly3[2]=-poly3[2];
}
    }
    p4[0]=l4*cos(angle);
    p4[1]=l4*sin(angle);
    alpha4=(pi/2)-angle;
    c=p4[1]/tan(alpha4);
    b=c+p4[0];
    a=b*tan(alpha4);
    //poly4=[a/b 1 a];
    poly4[0]=a/b;
    poly4[1]=1;
    poly4[2]=a;
    if (u2<0){
        poly4[0]=-poly4[0];
        poly4[1]=-poly4[1];
        poly4[2]=-poly4[2];
}
}

anglem = angle-dalpha; // Lower limit for next angle
anglep = angle+dalpha; // Upper limit for next angle
point1[0] = cos(anglem);
point1[1] = sin(anglem); // Calculate lower vector
point2[0] = cos(anglep);
point2[1] = sin(anglep); // Calculate upper vector


//lage begrensninger
if (l3>0){
    poly[0][0]=point1[1];
    poly[0][1]=-point1[0];
    poly[0][2]=0;
    poly[1][0]=-point2[1];
```

```
    poly[1][1]=point2[0];
    poly[1][2]=0;
    poly[2][0]=poly3[0];
    poly[2][1]=poly3[1];
    poly[2][2]=poly3[2];
    poly[3][0]=poly4[0];
    poly[3][1]=poly4[1];
    poly[3][2]=poly4[2];
    /*
    poly = [ point1(2) -point1(1) 0
    -point2(2) point2(1) 0
    poly3
    poly4];
     */
}else{
    poly[0][0]=point1[1];
    poly[0][1]=-point1[0];
    poly[0][2]=0;
    poly[1][0]=-point2[1];
    poly[1][1]=point2[0];
    poly[1][2]=0;
    poly[2][0]=0;
    poly[2][1]=0;
    poly[2][2]=0;
    poly[3][0]=poly4[0];
    poly[3][1]=poly4[1];
    poly[3][2]=poly4[2];
    /*
    poly = [ point1(2) -point1(1) 0
    -point2(2) point2(1) 0
    0 0 0
    poly4];
     */
}

A[0]=poly[0][0];
A[1]=poly[0][1];
A[2]=poly[1][0];
A[3]=poly[1][1];
A[4]=poly[2][0];
A[5]=poly[2][1];
A[6]=poly[3][0];
A[7]=poly[3][1];
```

```
but[0]=poly[0][2];
but[1]=poly[1][2];
but[2]=poly[2][2];
but[3]=poly[3][2];

//A=poly(:,[1 2]);
//b=poly(:,3);

}
```

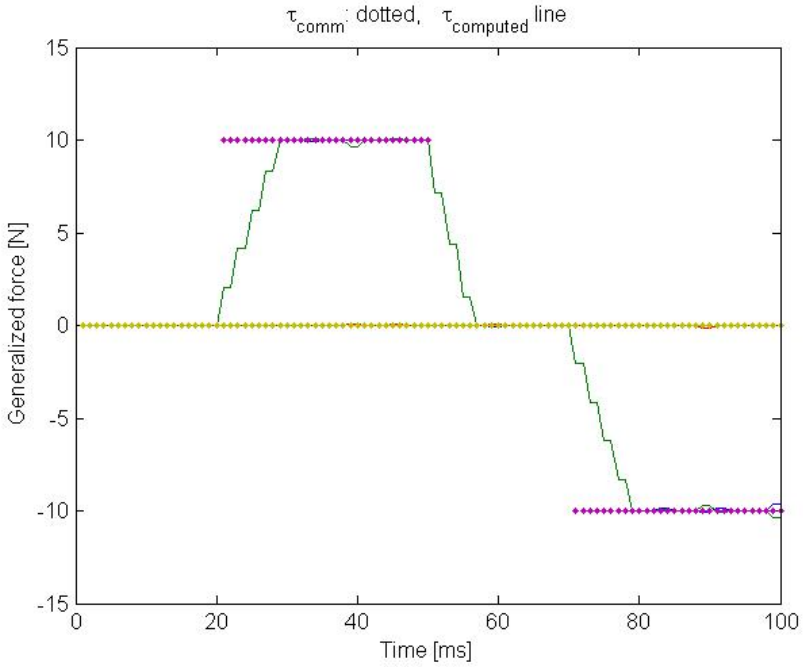
APPENDIX

B

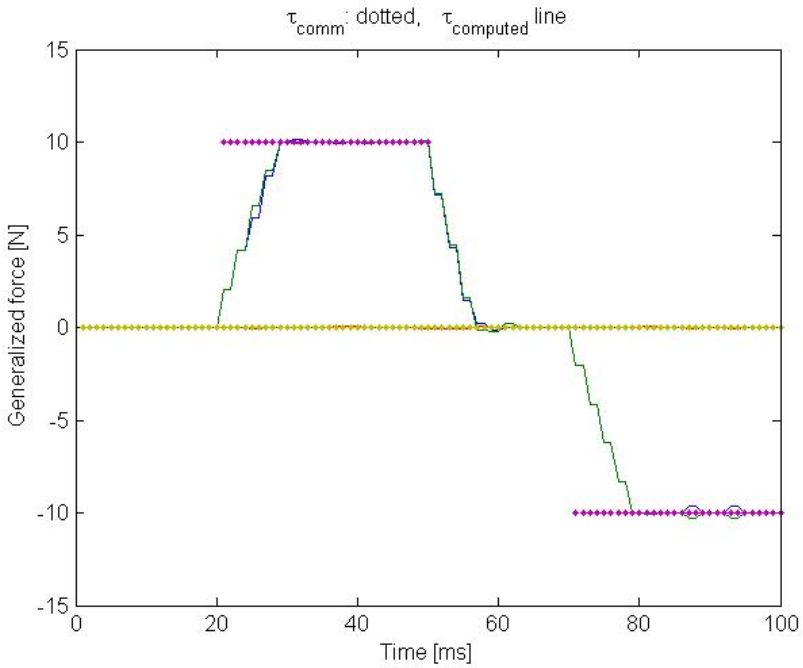
RESULTS OF SIMULATIONS

B.0.1 Algorithm 2

Fig. B.1 and Fig. B.2 shows the computed τ for each of the combinations, where $\boldsymbol{\tau}_{comm}$ is the same as in the former simulation. Fig. B.3 and Fig. B.4 shows the corresponding values of α. Fig. B.5 and Fig. B.6 the corresponding values of T. Fig. B.7 and Fig. B.8 is an illustration of the movements of the actuators in an xy-plot.

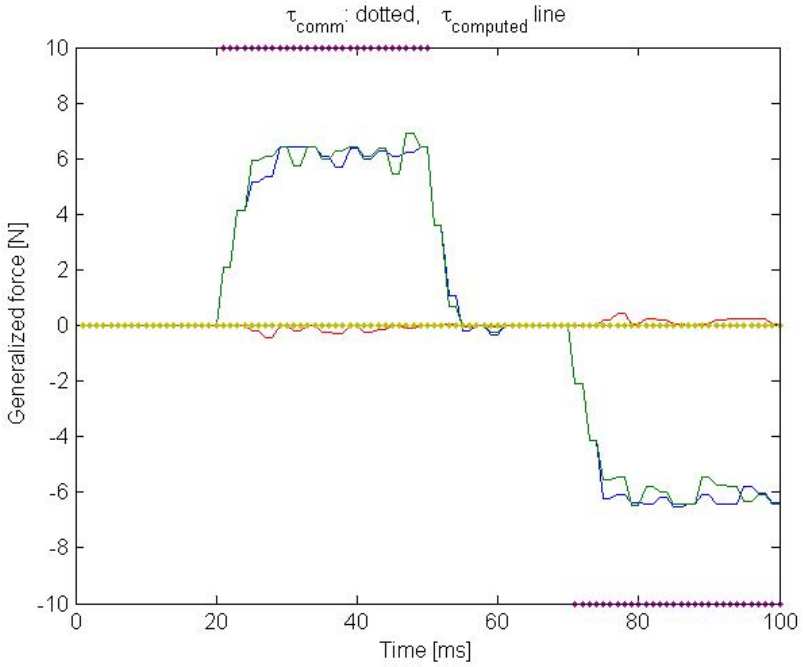


(a) Q_N=1, N=2

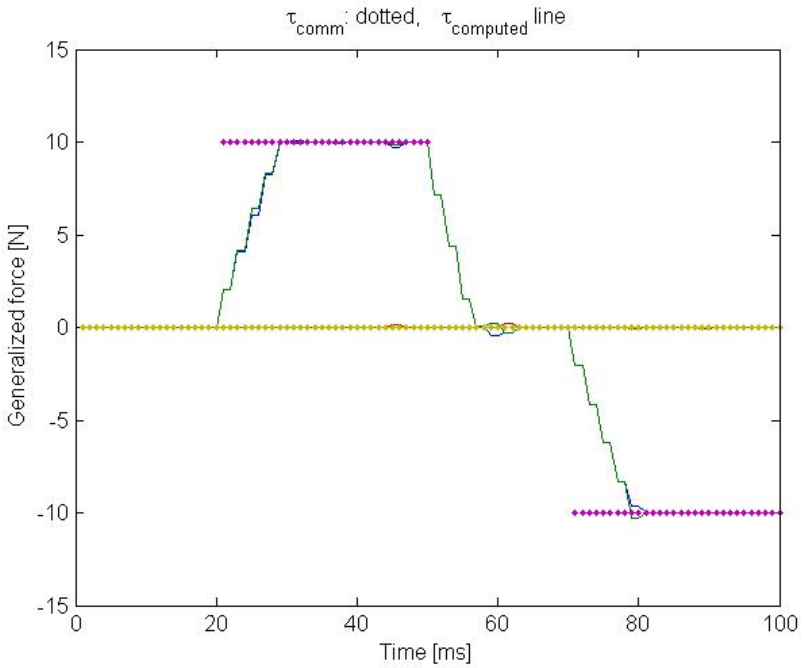


(b) Q_N=1, N=10

Figure B.1: Algorithm 2: τ for various horizon and weigths

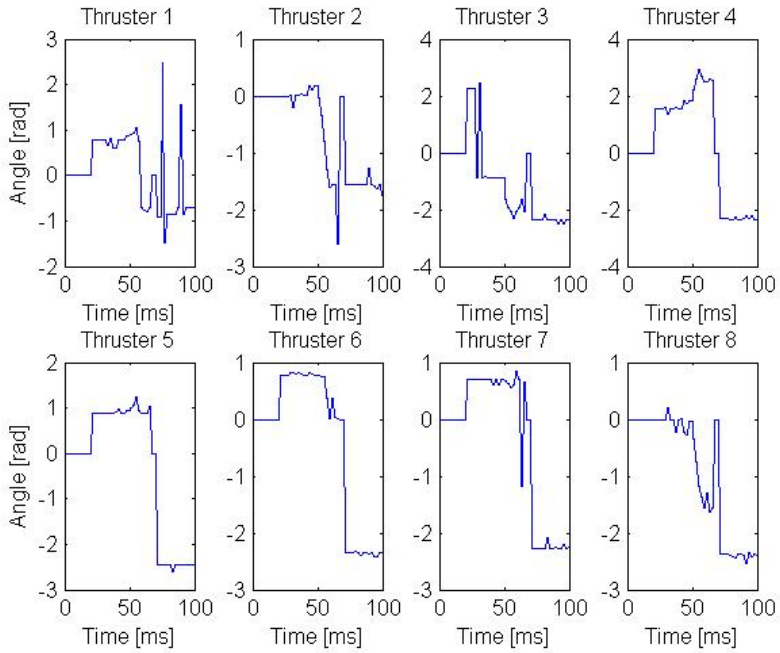


(a) Q_N=50, N=3

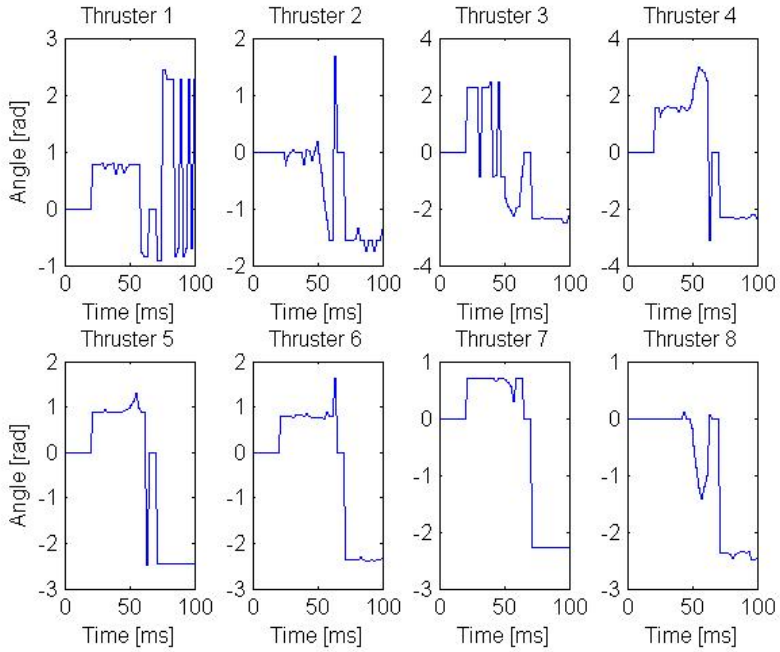


(b) Q_N=50, N=10

Figure B.2: Algorithm 2: τ for various horizon and weigths

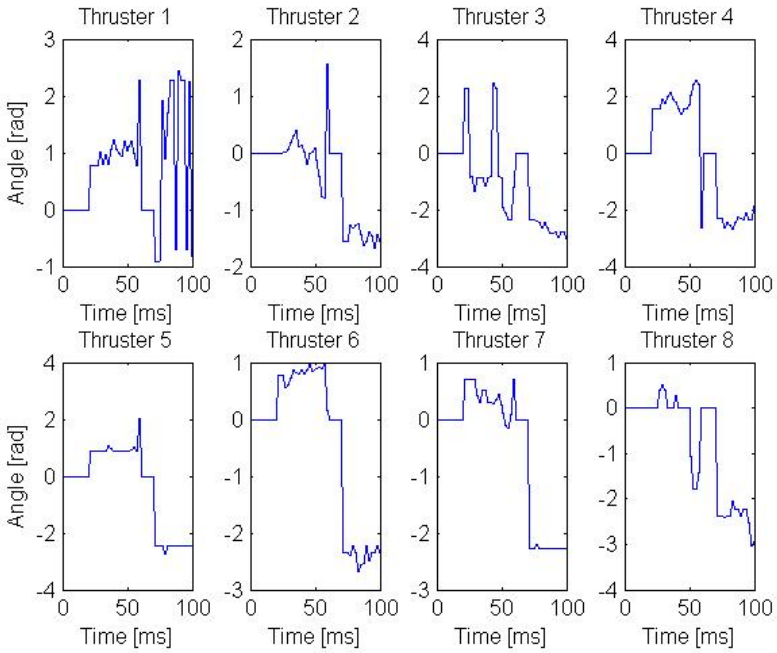


(a) Q_N=1, N=2

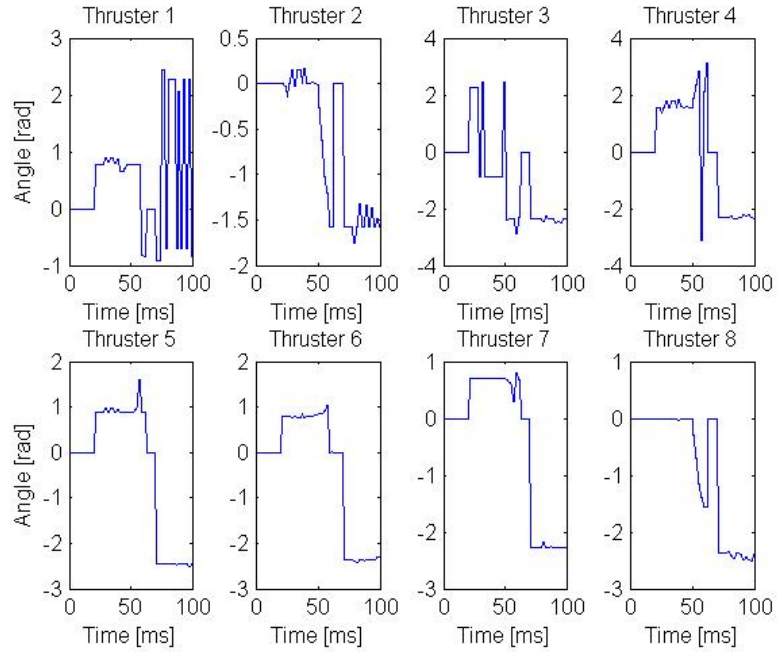


(b) Q_N=1, N=10

Figure B.3: Algorithm 2: α for various horizon and weigths

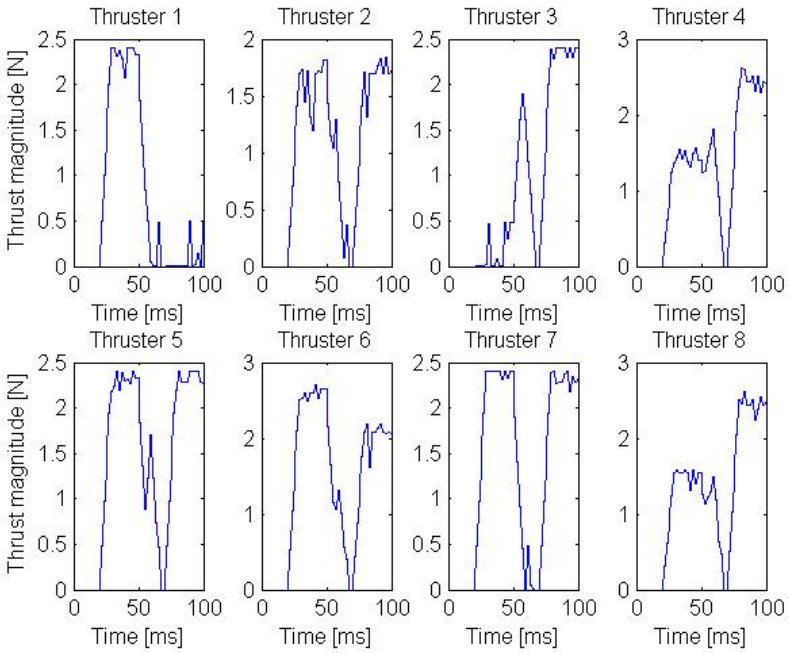


(a) Q_N=50, N=3

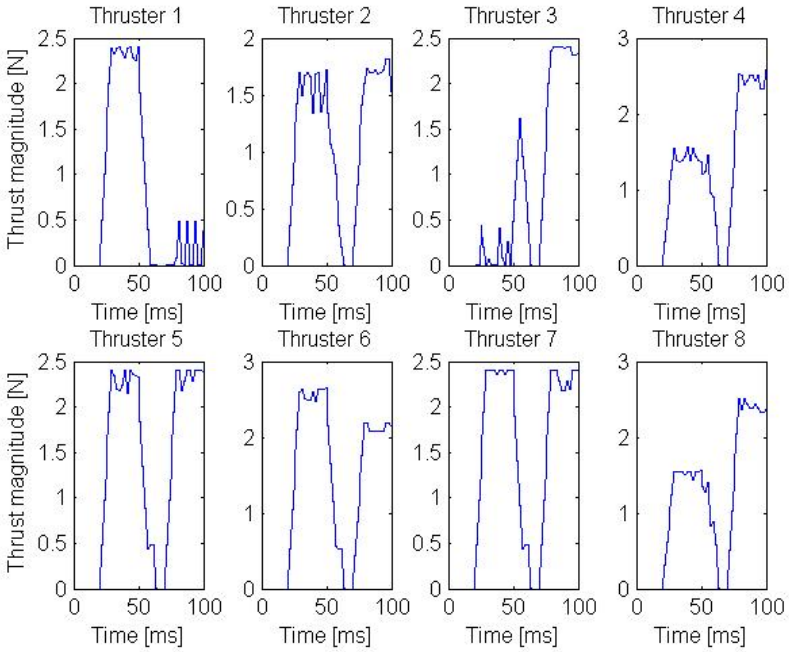


(b) Q_N=50, N=10

Figure B.4: Algorithm 2: α for various horizon and weigths

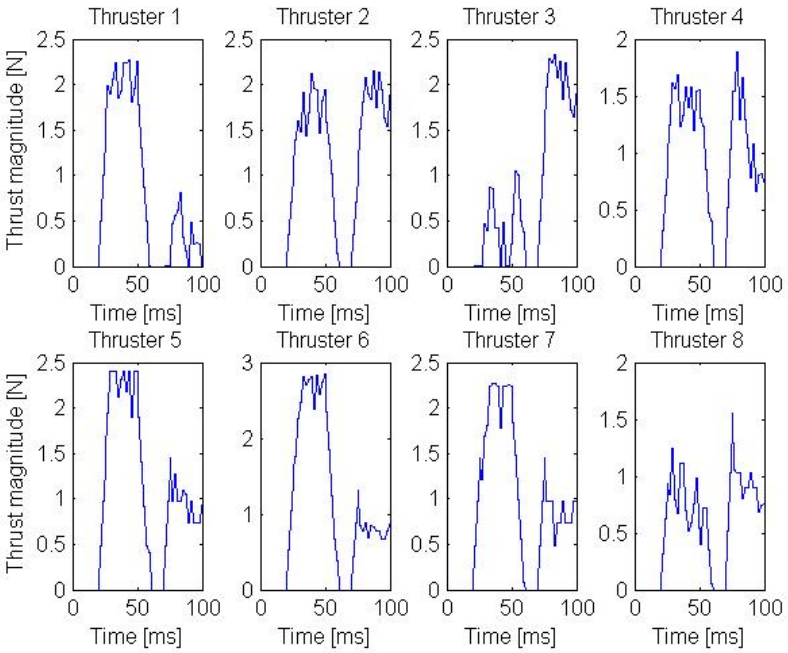


(a) Q_N=1, N=2

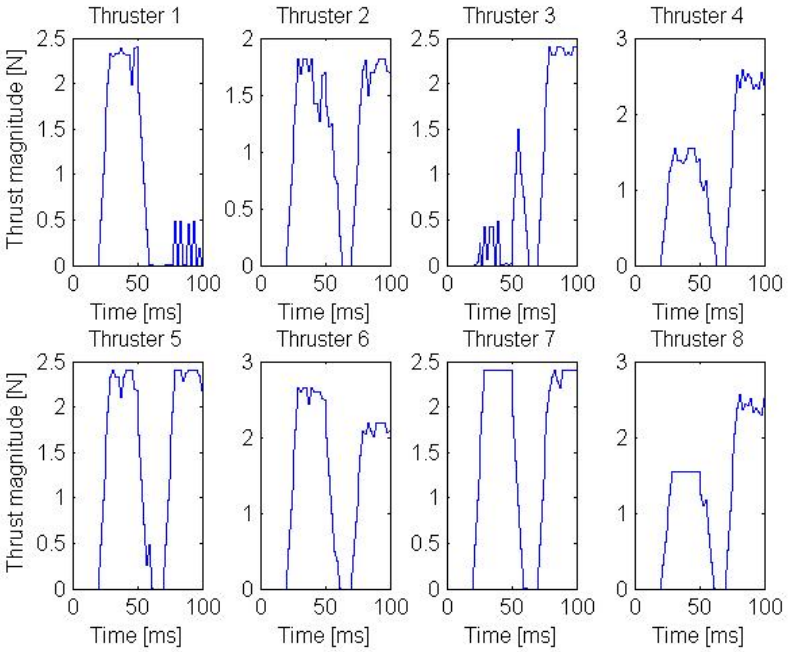


(b) Q_N=1, N=10

Figure B.5: Algorithm 2: T for various horizon and weigths

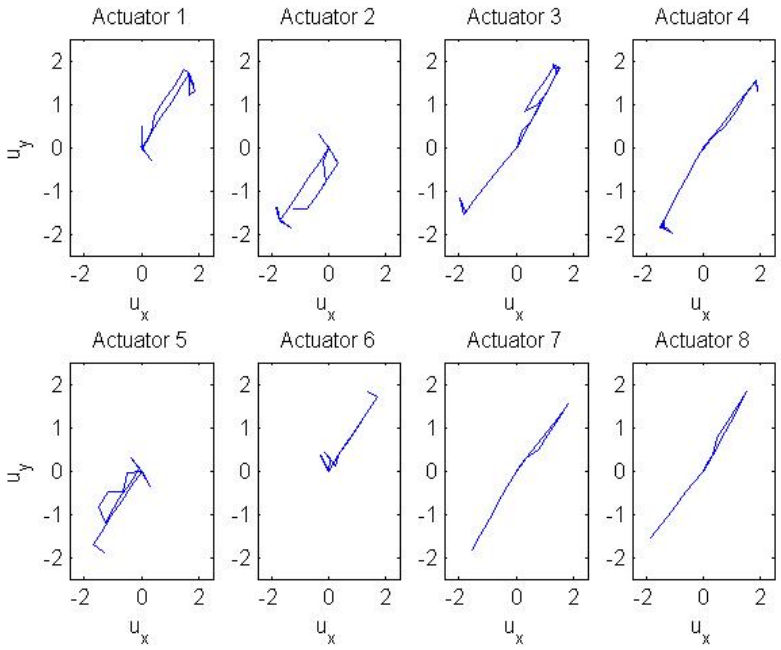


(a) Q_N=50, N=3

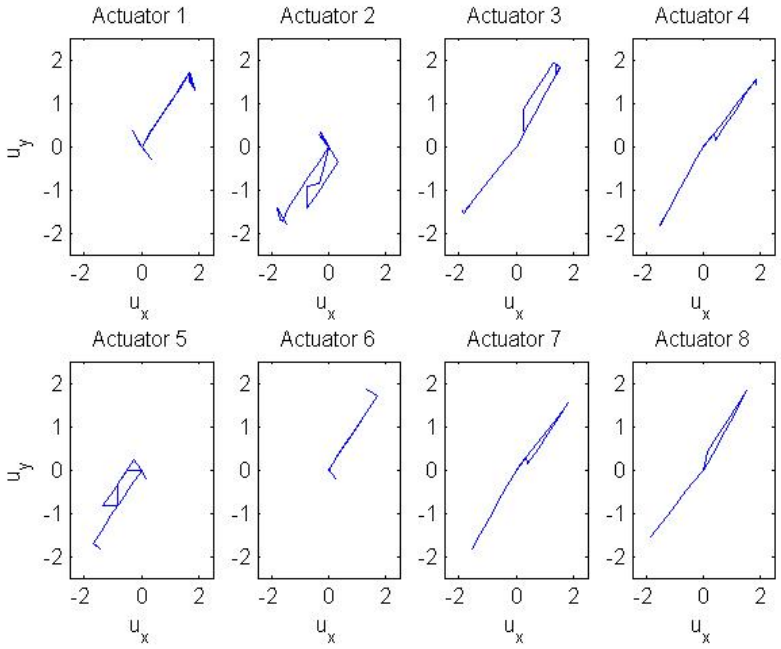


(b) Q_N=50, N=10

Figure B.6: Algorithm 2: T for various horizon and weigths

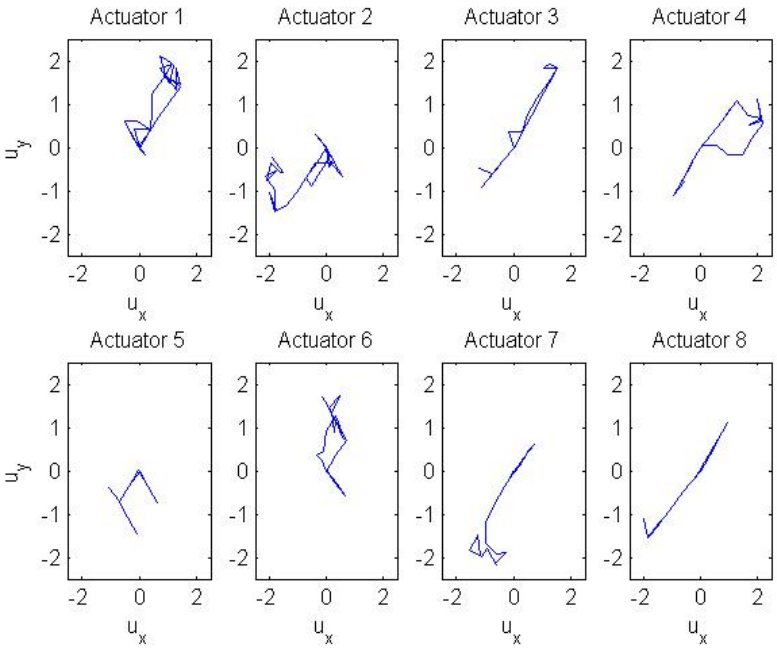


(a) Q_N=1, N=2

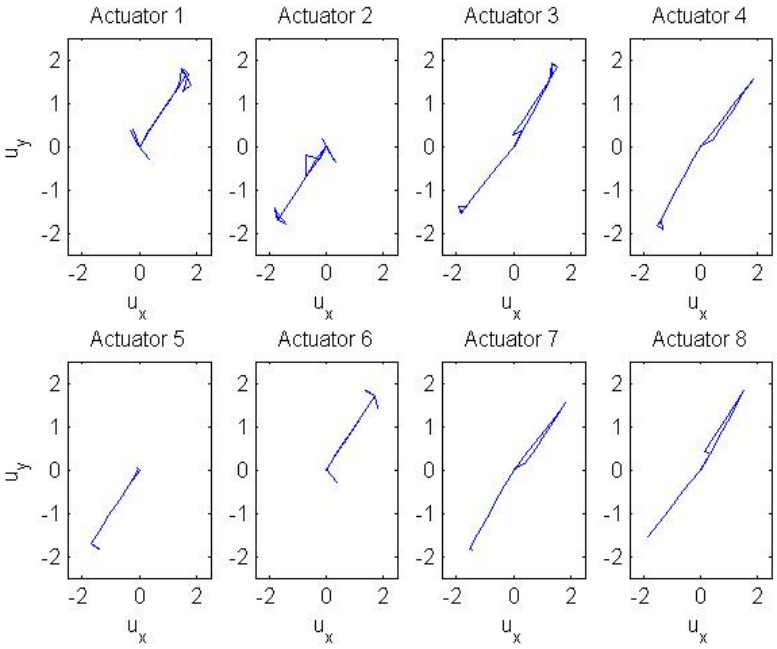


(b) Q_N=1, N=10

Figure B.7: Algorithm 2: Actuators for various horizon and weigths



(a) Q_N=50, N=3



(b) Q_N=50, N=10

Figure B.8: Algorithm 2: Actuators for various horizon and weigths

B.0.2 Algorithm 3

From simulations in Matlab, Fig. B.9 shows the resulting force for steps in $\boldsymbol{\tau}_{comm}$. Fig. B.10 and Fig. B.11 shows the corresponding α and T for the actuators, and Fig. B.12 how the force for each actuator 'move' around in the xy-plane.

Fig. B.0.2

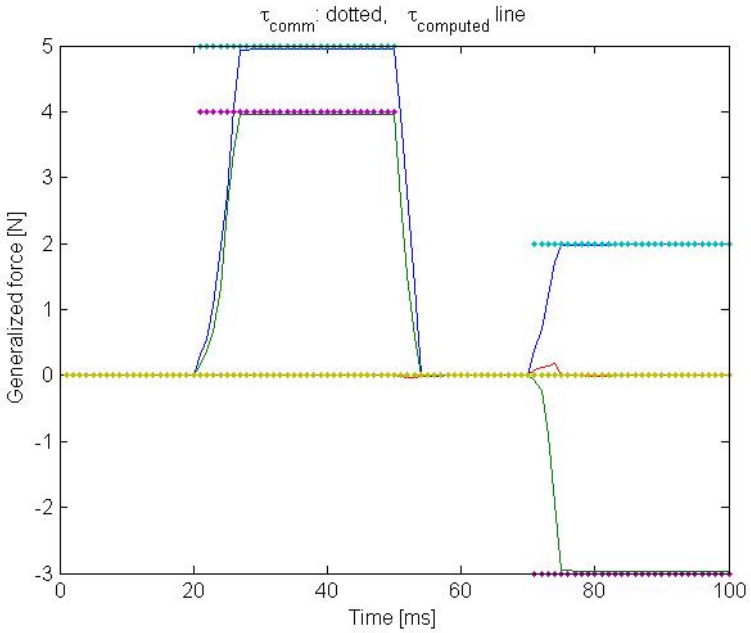


Figure B.9: Algorithm 3: Matlab simulation of force.

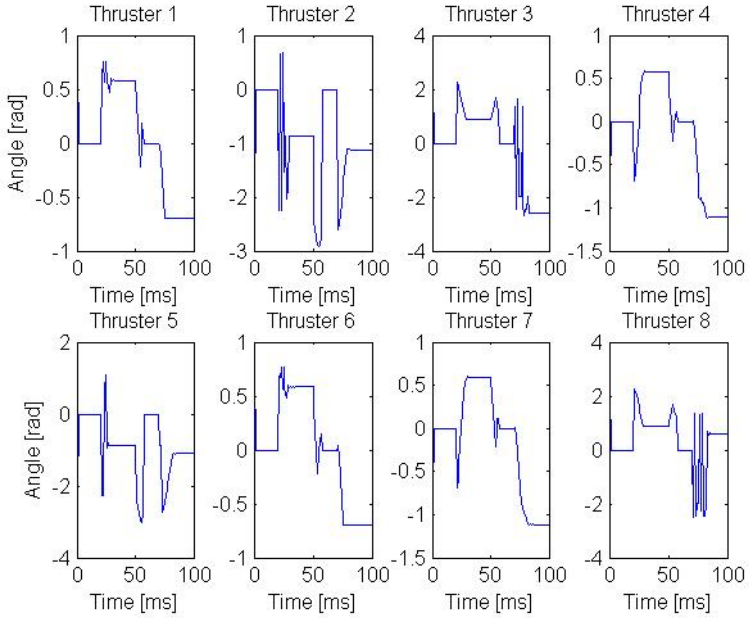


Figure B.10: Algorithm 3: Matlab simulation of thruster angle α

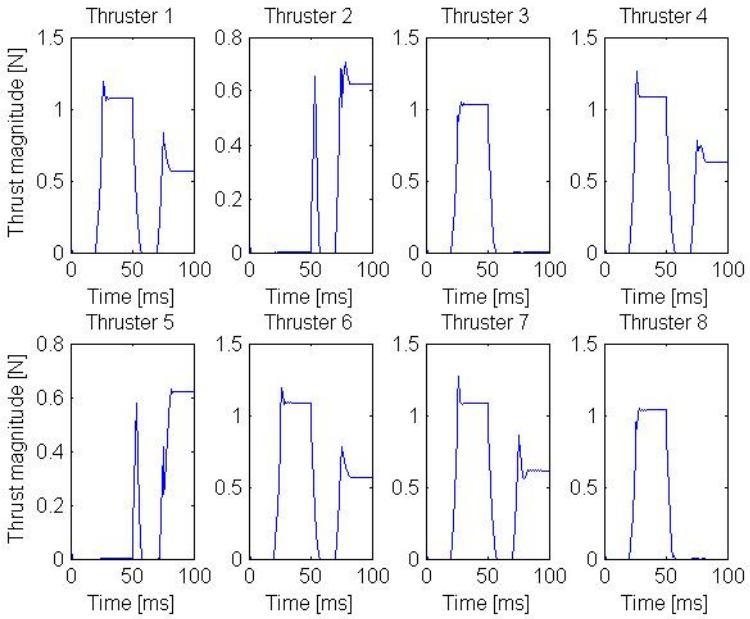


Figure B.11: Algorithm 3: Matlab simulation of thruster magnitude T

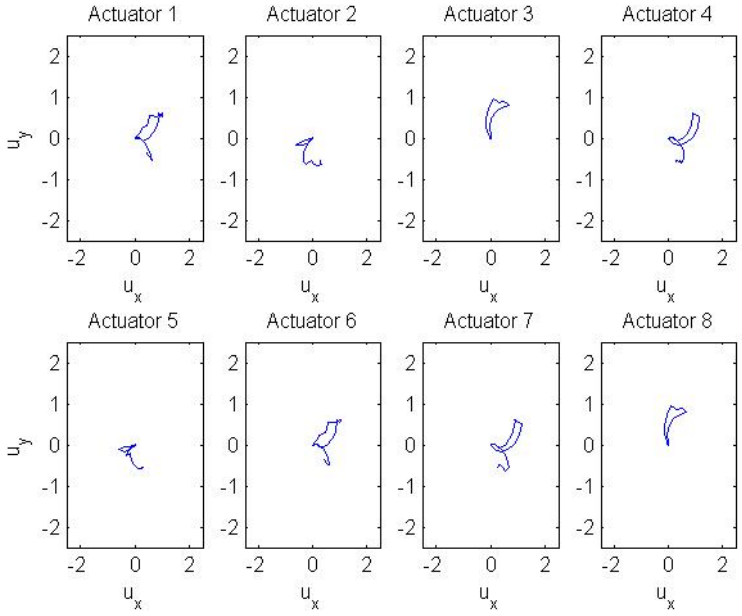


Figure B.12: Algorithm 3, Matlab simulation. Allowed sectors for the thrusters

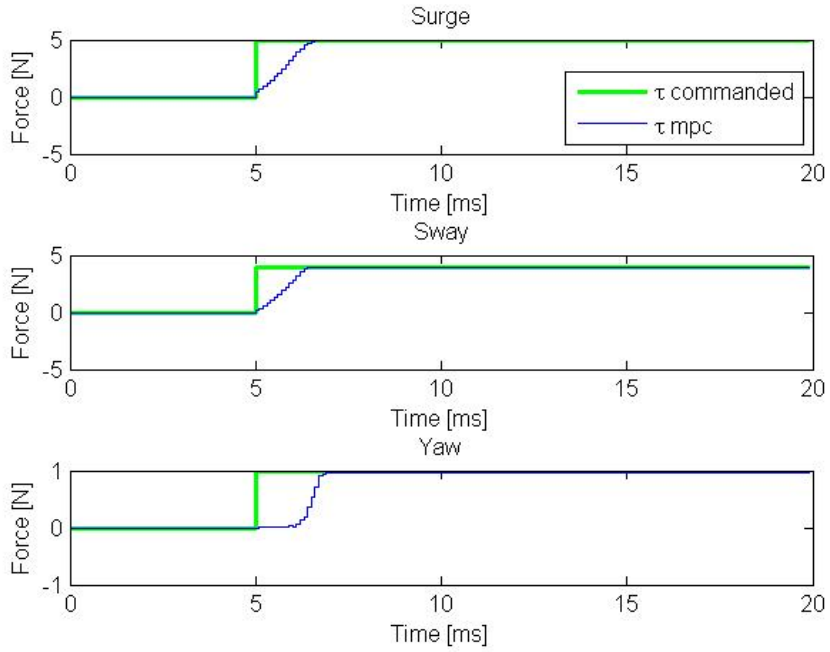


Figure B.13: Algorithm 3, implementation in C. Generalized force

APPENDIX

C

CYBERRIG I

CyberRig I (CRI) is a scaled replica of a semi-submersible drilling unit. It is equipped with two azimuth thrusters on each leg, eight in total. Each of the thrusters is controlled by a position-controlled step motor rotating the propeller housing, and a PRM-controlled motor driving the propeller.

Table C.2 shows relevant physical data for CyberRig I.

C.1 Hardware

The inboard micro PC of CyberRig I is an IEI Wafer-5822-300 PC/104 compatible board powered by a 300 MHz Pentium compatible CPU with 512 Mb RAM. It is running a QNX Neutrino Version 6.2 real time operating system, which is installed on a seperate 50 Gb hard drive, and controls 8 different cards connected to the PC/140 bus. The command station is a Dell LATTITUDE D800 laptop PC running Windows XP. It has a 1.6 GHz Pentium M processor and 512 Mb RAM. The micro PC placed on CyberRig I and the host PC communicate through a LAN.

The azimuth thruster propeller velocity is controlled by four Mesa Electronics 4127 Motor Controller Cards. 4127 is a low cost, LM629 based 2 axis DC servo motor control system implemented on a Stackable PC/104 card. Ther per axis output of the 4127 is an 8 bit sign-magnitude PWM signal that drives a Mesa 7127 dual H-Bridge intended for motion control applications.

Water density, ρ	1000
Propeller diameter, D	0.05 m
Thrust coefficient, K_T	0.6173
Sampling frequency	10 Hz
$l_{1,x}$	-0.305 m
$l_{1,y}$	0.365 m
$l_{2,x}$	-0.365 m
$l_{2,y}$	0.305 m
$l_{3,x}$	-0.365 m
$l_{3,y}$	-0.305 m
$l_{4,x}$	-0.305 m
$l_{4,y}$	-0.365 m
$l_{5,x}$	0.305 m
$l_{5,y}$	-0.365 m
$l_{6,x}$	0.365 m
$l_{6,y}$	-0.305 m
$l_{7,x}$	0.365 m
$l_{7,y}$	0.305 m
$l_{8,x}$	0.305 m
$l_{8,y}$	0.365 m
Upper limit on thrust magnitude, $\overline{T}_i$	2.4 N
Lower limit on thrust magnitude, $\underline{T}_i$	0 N

Table C.1: Physical data for CyberRig I

The rotation of the azimuth thrusters is controlled by three 5936 Stepper Motor Controller PC/104 Stackable cards. 5936 allows a PC/104 based computer sytem to control three independent SERVEX two phase stepper motor drivers. The orientation is read by a position sensor that outputs a analog votage between 0-5V to a DM6210 PC/104 compatible I/O card.

The case holding the eight cards of the PC/104 stack and the CPU is located in the middle of the rig connected to 4 boxes each containing a dual H-bridge and two stepper motor drivers. These boxes are connected to the corresponding cylinders numbered from 1 to 4, each containing two stepper motors for azimuth rotation and two RPM controlled propeller motors.

A conceptual schematics of the CyberRig I hardware is shown in Fig. C.1 according to Tyssø and Aga (2006).

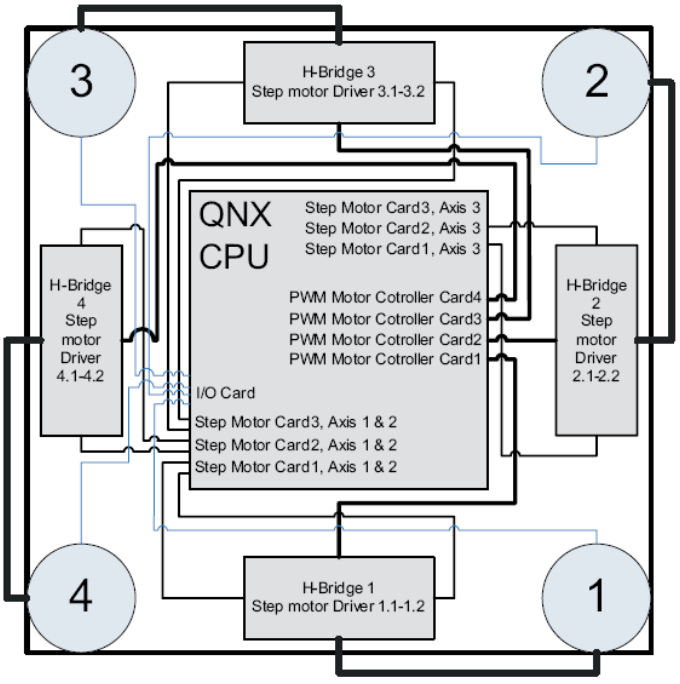


Figure C.1: Block diagram with control allocation block

The addresses of the motor driver cards and the I/O card on the CyberRig PC/104 bus are listed in Table C.2.

Card	*Address*
4127 Motor Controller Card 1	0x200
4127 Motor Controller Card 2	0x210
4127 Motor Controller Card 3	0x240
4127 Motor Controller Card 4	0x250
5936 Stepper Motor Card 1	0x300
5936 Stepper Motor Card 1	0x310
5936 Stepper Motor Card 1	0x320
DM 6210 I/O Card	0x350

Table C.2: CyberRig PC/104 address table

C.2 Software

The low-level software of CRI is written in C and implemented as s-functions in MATLAB/SIMULINK. The C code can be found in App. E. The SIMULINK models are handled by the Opal RT-lab which compiles all code necessary to download the model to the target QNX real-time system. The RT-Lab standard require the model to be divided into a console and a master subsystem. The console model (SC_Console) represent the user operated PC. The master console (SM_Master) represent the target QNX on CyberRig I.

C.2.1 quadprog vs. ql0001

In the implementation one has to be avare of that the quadratic programming solvers *quadprog* in matlab and *ql001* in c works slightly different. In *quadprog* the following problem is minimized:

$$J = \min_x \frac{1}{2} x^T H x + f^T x \tag{C.1}$$

subject to:

$$\begin{aligned} Mx &\leq n \\ M_{eq} x &= n_{eq} \\ lb &\leq x \leq ub \end{aligned} \tag{C.2}$$

In ql001 on minimize the following:

$$J = \min_x \frac{1}{2} x^T C x + D^T x \tag{C.3}$$

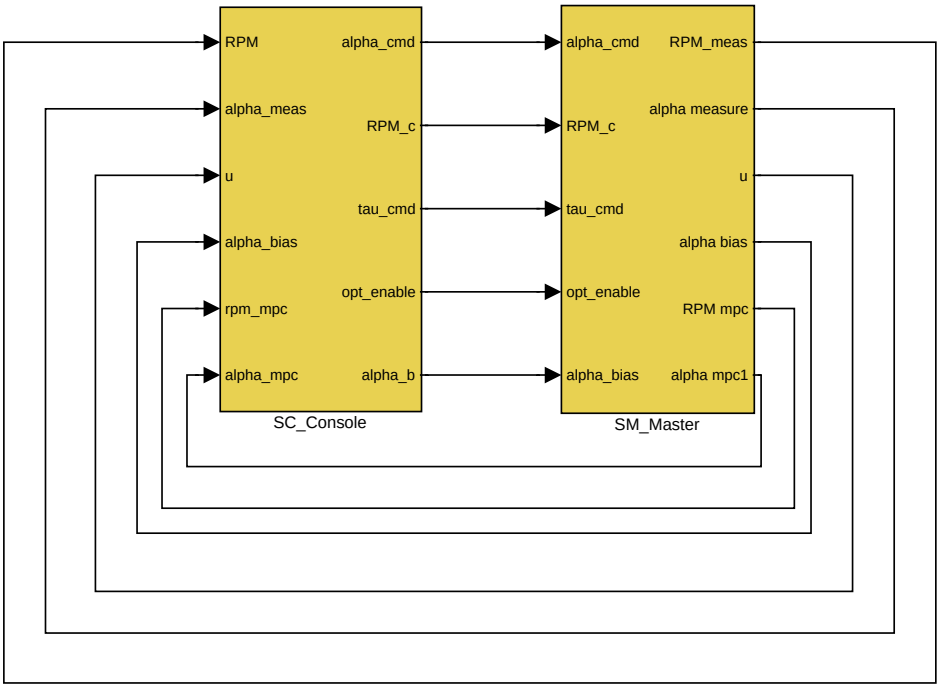


Figure C.2: The console and master subsystems

subject to:

$$\begin{aligned} &A(J)x + B(J) = 0, \quad J = 1, \ldots, ME \\ &A(J)x + B(J) \geq 0, \quad J = ME+1, \ldots, M \\ &xl \leq x \leq xu \end{aligned} \tag{C.4}$$

It is reqognized that $C = H$, $D = f$, $xl = lb$, $xu = ub$. But for the first two constraints some signs must be changed, noticing that:

$$\begin{aligned} -M_{eq}x + n_{eq} &= 0 \\ -Mx + n &\geq 0 \end{aligned}$$

Then we have that

$$\begin{aligned} A(J) = -M_{eq}, \; B(J) = n_{eq}, \quad J = 1, \ldots, ME \\ A(J) = -M, \; B(J) = n, \quad J = ME+1, \ldots, M \end{aligned}$$

Also when using the function ql001 one should be avare of that matrices are saved as in fortran: row major order. In c it is saved in column major order.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \Rightarrow \begin{array}{c} row\ major\ order \longrightarrow 1\ 2\ 3\ 4\ 5\ 6 \\ column\ major\ order \longrightarrow 1\ 4\ 2\ 5\ 3\ 6 \end{array} \tag{C.5}$$

Therefore, while working with multiple arrays in c (and S-function Builder), the matrices should be transposed before they are used by ql001. This is done using the function c2f(...).

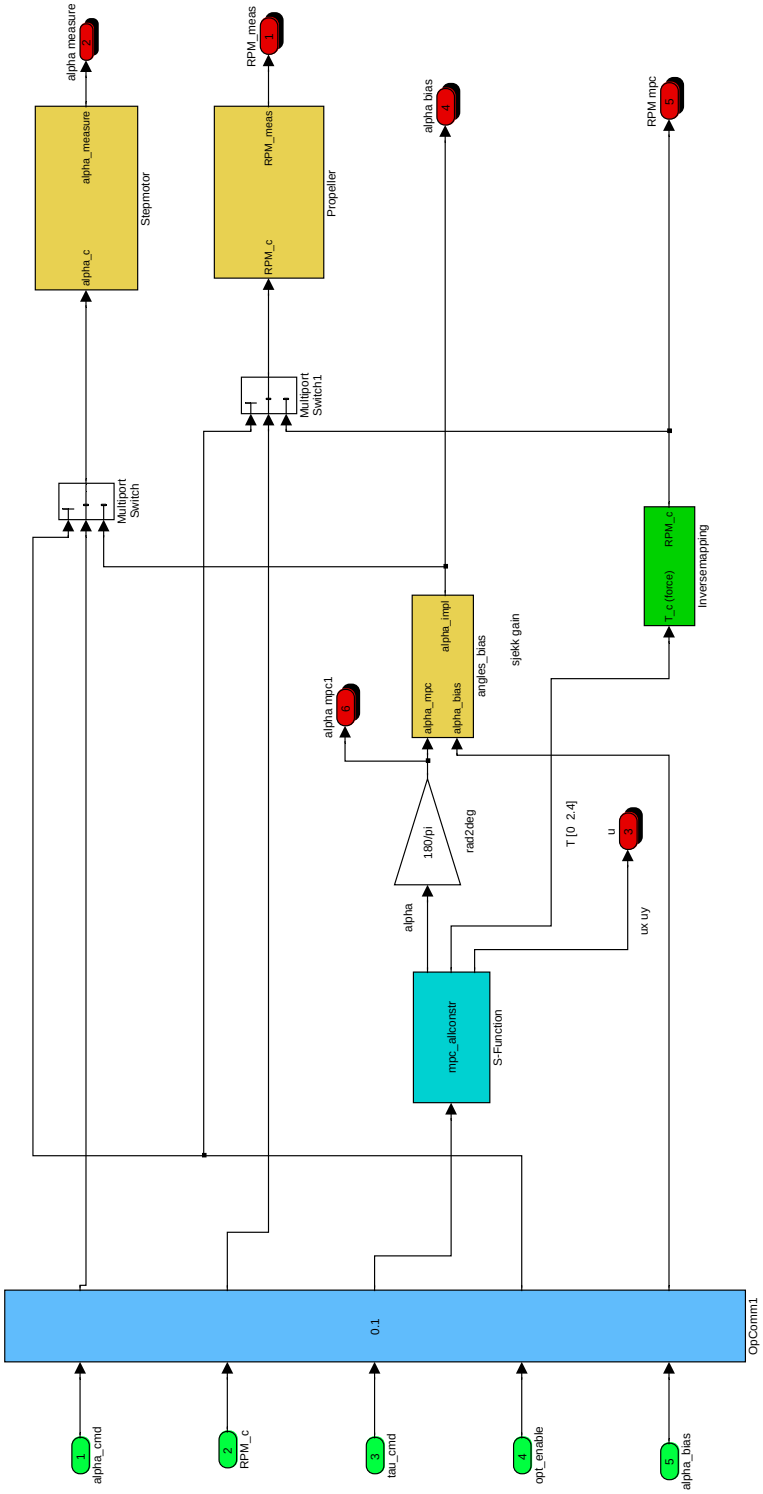


Figure C.3: The master subsystem

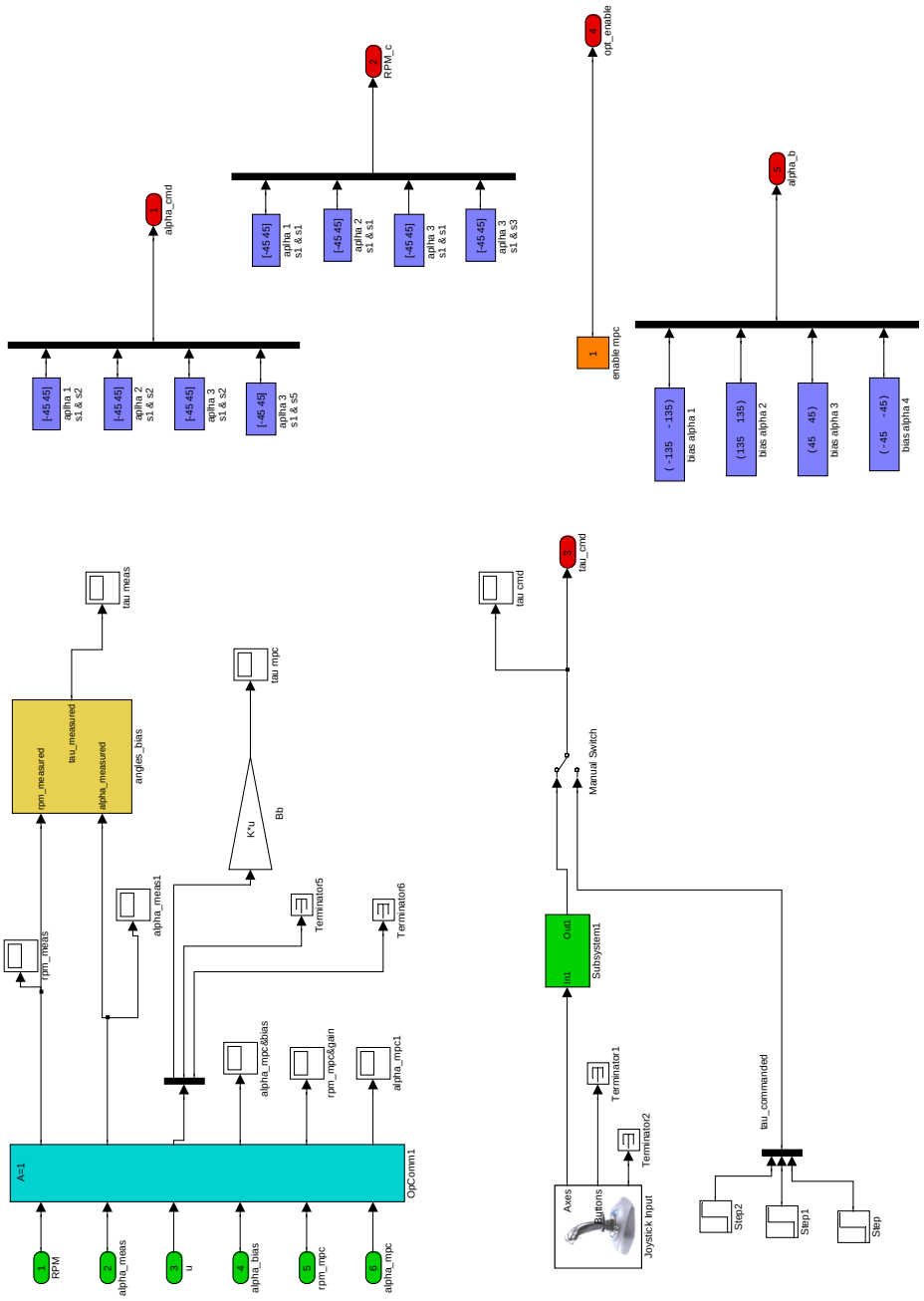


Figure C.4: The console subsystem

APPENDIX

D

MCLAB

The Marine Cybernetics Laboratory (MCLab) is an experimental laboratory for testing of ships, rigs, underwater vehicles and propulsion systems. As the name indicates, the facility is especially suited for tests of marine control systems, due to the relatively small size and advanced instrumentation package. It is also suitable for more specialised hydrodynamic tests, mainly due to the advanced towing carriage , which has capability for precise movement of models in 6 degrees of freedom.

The laboratory was appointed as an Marie Curie EU Training Site in 2002. The laboratory is a joint facility between Department of Engineering Cybernetics at NTNU, Department of Marine Technology at NTNU and Marintek (the Norwegian Marine Technology Research Institute).

Capacities:

- Tank Dimensions: L x B x D = 40m x 6.45m x 1.5m
- Wave generator : Hs = 0.3m, T=0.6-1.5s (irregular waves)
- Carriage : towing speed 2 m/s, 5 (6) DOFs forced motions
- Current generation: 0-0.15m/s
- Computer system for control, data recording and analysis
- Typical scaling ratios: l = 50-150
- Typical ship model lengths:1-3m

A picture of CyberRig I in MCLab is shown in Fig. D

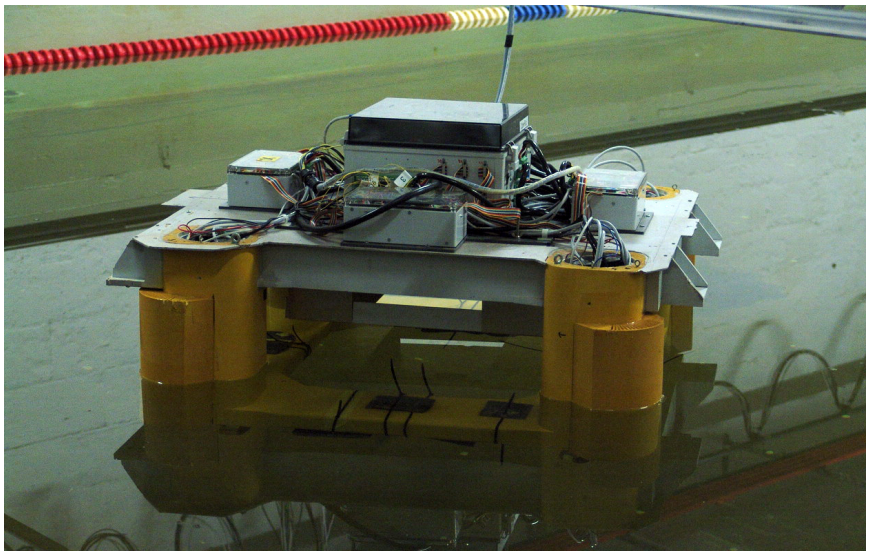


Figure D.1: CyberRig I in MCLab

For more information about the lab, please visit

`http://www.itk.ntnu.no/marinkyb/MCLab/index.html`.

APPENDIX

E

DIGITAL ATTACHMENT

The attached zip-file contains:

- This report
- Code for Matlab-simulations
- Simulink models and control-allocation algorithms written in C, implemented as Matlab S-functions