NTNU
Norwegian University of
Science and Technology

Control Allocation for Underwater Snake Robots using Optimization Methods

Siri Bjørkedal Øvregård

Marine Technology
Submission date: June 2018
Supervisor: Dong Trong Nguyen, IMT
Co-supervisor: Asgeir Sørensen, IMT
Henrik Schmidt-Didlaukies, IMT

Norwegian University of Science and Technology
Department of Marine Technology

Preface

This is a master's thesis written during the spring of 2018 at Norwegian University of Science and Technology (NTNU), Department of Marine Technology. This thesis concerns thrust allocation for underwater snake robots, and simulations are done in Matlab/SIMULINK using an underwater snake robot simulation model. Results are given in the thesis.

Although I have previously studied underwater snake robots, the exact idea for the topic of this thesis was given to me by my supervisor Dong Trong Nguyen, and the problem description was formulated in collaboration with him and my co-supervisor Henrik Schmidt-Didlaukies.

For reading and understanding this report, no extensive knowledge of marine technology or control theory is needed as all subjects will be properly introduced and described throughout the text. However, general engineering knowledge is an advantage.

Trondheim, 2018-06-25

Siri Bjørkedal Øvregård

Summary

Although the concept of underwater vehicles has existed for a very long time, the technology within the field has progressed substantially over the recent decades. As more and more oil and gas installations and operations are performed subsea, the need for more developed underwater vehicles is still present. A new concept which has the potential to fulfill these demands is the Underwater Underwater Snake Robot. The fact that the robot is shaped as a snake makes it ideal for moving in high viscosity environments such as water. In addition, the robot's ability to alter configuration provides a large workspace, and its slender and articulate body allows the robot to access narrow spaces. The fact that the underwater snake robot is, itself, a manipulator arm capable of performing light intervention tasks, makes it a powerful tool.

In this thesis, methods for thrust allocation for underwater snake robots are evaluated. Two iterative methods, using linear and quadratic programming, are presented, developed and implemented. This is also done for an explicit method for constrained thrust allocation, using redistributed pseudo-inverse. The methods are implemented into an existing underwater snake robot simulation model in Matlab/SIMULINK. Simulations are performed for unconstrained and constrained thrust allocation, simulating planar and thee-dimensional motion. In the unconstrained case, simulations are also performed using a pre-implemented standard damped inverse algorithm. This is done in order to compare thrust allocation algorithm performances.

It is found that all developed algorithms produce satisfactory simulation results, although some variations in performance is found. The linear programming algorithm produces a small error between commanded and actual thrust, but tends to favor using a low amount of thrusters, which is not ideal. In the constrained case, the performance of this algorithm is better.

The redistributed pseudo-inverse algorithm produces a large error compared to the other methods. The performance is therefore found to be sub-optimal. The quadratic programming algorithm produces low errors for all simulation cases. The algorithm also tends to distribute the commanded thrust more evenly amongst the thrusters. This is a significant up-side as it reduces wear and tear on the thrusters.

Thus, it is concluded that the quadratic programming algorithm for thrust allocation produces the most satisfactory results, although all thrust allocation methods have proven to be viable for use on underwater snake robots.

Samandrag

Sjølv om undervassfartøy-konseptet har eksistert i svært lang tid, har teknologien innan feltet utvikla seg kraftig først dei siste tiåra. Etter kvart som fleire og fleire olje- og gassinstallasjonar og operasjonar utførast subsea, er behovet for meir utvikla undervassfartøy framleis tilstades. Eit nytt konsept som har potensiale til å oppfylle desse krava er undervass-slangeroboten. Det faktum at roboten er forma som ein slange gjer den ideell til å bevege seg i miljø med høg viskositet, slik som vatn. I tillegg gjer roboten si evne til å endre konfigurasjon at den har eit stort arbeidsområde, og den slanke og artikulerte kroppen gjer det mogeleg for roboten å få tilgang til smale passasjar. Det faktum at undervass-slangeroboten i seg sjølv er ein manipulatorarm, noko som gjer den istand til å utføre lette intervensjonsoppgåver, gjer den til eit kraftig verktøy.

I denne oppgåve blir metodar for thrustallokering for undervass-slangerobotar vurdert. To iterative metodar, *lineær* og *kvadratisk programmering*, er presentert og implementert. Det same gjeld for *omfordelt pseudo-invers*, ein eksplisitt metode for begrensa thrustallokering. Metodane blir implementert i ein eksisterande simuleringsmodell for undervass-slangerobotar ved bruk av Matlab/SIMULINK. Simuleringane blir utført for både ubegrensa og begrensa thrustallokering, der både plan bevegelse og tre-dimensjonal bevegelse blir simulert. I tilfellet med ubegrensa thrustallokering blir det også utført simuleringar med ein pre-implementert *standard dempa invers* algorithme. Dette blir gjort for å kunne samanlikne og vurdere ytelsen til dei forskjellige thrustallokeringsalgoritmane.

Det blir funne at alle dei implementerte algoritmane gir tilfredsstillande simuleringsresultat, sjølv om ytelsen frameleis varierer noko. Den lineære programmeringsalgoritmen produserer ein lav verdi for feilen mellom thrust-kommando og faktisk thrust, men har ein tendens til å favorisere bruk av færre thrusterar, noko som ikkje er ideelt. I tilfellet med begrensa thrustallokering er ytelsen noko betre.

Den omfordelte pseudo-invers algoritma gir ein stor feilverdi samanlikna med dei andre metodane. Resultatet kan derfor seiast å vere sub-optimalt. Algoritma med kvadratisk programmering produserer derimot lave feilverdiar ved alle simuleringar. Algoritmen har også ein tendens til å distribuere thrust-kommando krefter meir jevnt mellom thrusterane. Dette er eit signifikant fortrinn med metoden, sidan det fører til mindre slitasje på enkelte trusterar.

Det kan dermed konkluderast at algorithmen som brukar kvadratisk programmering gir dei mest tilfredsstillande resultata, sjølv om alle thrustallokeringsmetodane har vist seg å vere levedyktige for bruk på undervass-slangerobotar.

Acknowledgment

I would like to thank my supervisor Dong Trong Nguyen for all the support and feedback he has given me during the work with this thesis. Our frequent meetings and communication has helped me gain the knowledge I needed to perform my work. I would also like to thank Jørgen Sverdrup-Thygeson and Ida-Louise Garmann Borlaug for the hours they have spent giving me insight about the underwater snake robot simulation model.

My utmost gratitude is extended to my co-supervisor Henrik Schmidt-Didlaukies for always being available, for answering all my questions, and for being a source of motivation.

And finally, I would like to thank my friends and family for their invaluable and unconditional support.

S.B.Ø.

Nomenclature and Abbreviations

The following lists describes several abbreviations and symbols that will be used within the body of this thesis. For mathematical symbols, scalars are generally given by normal lowercase (a) or uppercase (A) letters, while vectors are given by boldface lowercase letters (**b**) and matrices are given by boldface uppercase letters (**B**).

Abbreviations

AUV	Autonomous Underwater Vehicle
DOF	Degrees of Freedom
IMR	Inspection, Maintenance and Repair
LP	Linear Programming
QP	Quadratic Programming
ROV	Remotely Operated Vehicle
RPI	Redistributed Pseudo-Inverse
RSI	Redistributed Pseudo-Inverse
SDI	Standard Damped Inverse
SVD	Singular Value Decomposition
USM	Underwater Swimming Manipulator
USR	Underwater Snake Robot
UVMS	Underwater Vehicle-Manipulator Systems

Nomenclature

$\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{q} \end{bmatrix}^T$	Position and orientation of the base frame in inertial frame

$\boldsymbol{\gamma}_0$	Vector giving direction of gravity in inertial frame
$\boldsymbol{\gamma}_i$	Unit vector giving direction of gravity for link i
$\boldsymbol{\nu}$	Body velocity vector of base frame
$\boldsymbol{\omega}^b$	Body-fixed angular velocity vector
$\boldsymbol{\Sigma}$	Diagonal singular value matrix
$\boldsymbol{\tau}_c$	Commanded thrust
$\boldsymbol{\tau}_q$	Joint torque vector
$\boldsymbol{\theta}$	Joint angle vector
$\boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\nu}^T & \dot{\boldsymbol{\theta}}^T \end{bmatrix}^T$	Global body velocity vector
$\boldsymbol{A}_i(\theta_i)$	Transformation matrix for transformation between link $i+1$ frame and link i frame
$\boldsymbol{B}$	Thrust configuration matrix
$\boldsymbol{B}^\dagger$	Moore-Penrose pseudo-inverse
$\boldsymbol{B}_a$	Thrust configuration matrix for angular motion
$\boldsymbol{B}_l$	Thrust configuration matrix for linear motion
$\boldsymbol{C}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta}$	Centripetal and coriolis effects
$\boldsymbol{C}_\varepsilon$	Standard damped inverse
$\boldsymbol{C}_a$	Standard damped inverse for angular motion
$\boldsymbol{C}_l$	Standard damped inverse for linear motion
$\boldsymbol{D}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta}$	Drag effects
$\boldsymbol{D}_{L,i}$	Linear drag force contribution
$\boldsymbol{D}_{NL,i}(\boldsymbol{\theta},\boldsymbol{\zeta})$	Nonlinear drag force contribution
$\boldsymbol{g}(\boldsymbol{\eta},\boldsymbol{\theta})$	Hydrostatic forces
$\boldsymbol{H}_i$	Special Euclidean group for link i

$\boldsymbol{J}$	Jacobian matrix
$\boldsymbol{J}_s$	Cost function
$\boldsymbol{M}(\boldsymbol{\theta})\dot{\boldsymbol{\zeta}}$	Mass forces
$\boldsymbol{M}_i$	Total mass of link i
$\boldsymbol{M}_{A,i}$	Added mass of link i
$\boldsymbol{M}_{R,i}$	Rigid body mass of link i
$\boldsymbol{q} = \begin{bmatrix} \eta \\ \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix}$	Unit quaternions
$\boldsymbol{q}$	Error weight vector
$\boldsymbol{R}$	Rotation matrix
$\boldsymbol{R}_i$	Rotation matrix for link i
$\boldsymbol{R}_{\theta_i}$	Euler angle rotation matrix
$\boldsymbol{r}_{b,i}$	Location of the centre of buoyancy of link i
$\boldsymbol{R}_b^n(\boldsymbol{q})$	Quaternion transformation matrix between base and inertial frame
$\boldsymbol{r}_{g,i}$	Location of the centre of gravity of link i in the local link frame
$\boldsymbol{SE}(3)$	Special Euclidean group of order 3
$\boldsymbol{SO}(3)$	Special orthogonal group of order 3
$\boldsymbol{S}(\cdot)$	Skew-symmetric matrix
$\boldsymbol{s}_{max}$	Maximum error vector
$\boldsymbol{t}$	Translation vector
$\boldsymbol{T}(\boldsymbol{\eta})$	Transformation matrix for all motion between base and inertial frame
$\boldsymbol{T}(\boldsymbol{H}_i)$	Transformation matrix for transformation between link i frame and inertial frame
$\boldsymbol{t}_i$	Translation vector for link i

$\boldsymbol{T}_q(\boldsymbol{q})$	Transformation matrix for angular velocities between base and inertial frame
$\boldsymbol{U}$	Left unitary matrix
$\boldsymbol{u}$	Thrust vector
$\boldsymbol{u}_{max}$	Maximum thrust vector
$\boldsymbol{u}_{min}$	Minimum thrust vector
$\boldsymbol{V}$	Rights unitary matrix
$\boldsymbol{v}^b$	Body-fixed velocity vector
$\boldsymbol{w}$	Thrust weight vector
$\boldsymbol{W}_a$	Diagonal matrix with weights for angular motion
$\boldsymbol{W}_l$	Diagonal matrix with weights for linear motion
$\dot{\boldsymbol{\eta}}$	Velocity vector of base frame in inertial frame
$\dot{\boldsymbol{\theta}}$	Joint angle velocity vector
$\dot{\boldsymbol{J}}_i$	Time derivative of the Jacobian
$\dot{\boldsymbol{p}}^n$	Inertial frame (NED) velocity vector
$\dot{\boldsymbol{q}}$	Angular body velocity in inertial frame
γ	Boundary limit
ρ	Density of water
σ_i	Singular value
θ_i	Individual joint angle
ε	Damping constant
$\{b\}$	BODY (Body-Fixed Reference Frame)
$\{n\}$	NED (Earth-Fixed Reference Frame)
C_a	Added mass coefficient
$C_{d,i}$	Drag coefficient

g	Gravitational acceleration constant
$I_{R,i}$	Rigid body inertia matrix of link i
l_i	Length of link i
m	Number of forces and moments equal to the number of degrees of freedom in a system
m_i	Mass of link i
n	Number of links
p	Number of thrusters
r	Radius of links
s_i^+, s_i^-, u_i^+, u_i^-	Auxiliary variables
V_i	Volume of link i
v_{ref}	Reference velocity

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Although the concept of underwater vehicles has existed for a very long time, the technology within the field has progressed substantially over the recent decades. Both Remotely Operated Vehicles (ROVs) and Autonomous Underwater Vehicles (AUVs) exist today in a vast variety of shapes and forms and operational functionality. ROVs equipped with manipulator arms, often referred to as Underwater Vehicle-Manipulator Systems (UVMS), have been essential for development and maintenance of subsea installations. However, as more and more oil and gas installations and operations are performed subsea, the need for even more developed underwater vehicles that can perform increasingly complex tasks and operations such as installation support and inspection, maintenance and repair (IMR) is still present.

A relatively new concept, which has the potential to fulfill these demands, is the Underwater Swimming Manipulator (USM), which is also referred to as the Underwater Snake Robot (USR). The USR is an AUV with the properties of an UVSM where the snake itself is a manipulator arm with a floating base. The fact that the robot is shaped as a snake makes it ideal for moving in high viscosity environments such as water. In addition, the robot's ability to alter configuration provides a large workspace, and its slender and articulate body allows the robot to access narrow spaces.

The USR is made up of links interconnected by actuated joints. Research on snake robot locomotion has previously been concentrated on locomotion forms similar to the ones found for biological snakes. These locomotion forms are produced by joint actuation, which allows for the snake robot to move silently and with minimal power consumption. However, for the USR as a UVMS, the biological snake locomotion forms require a body configuration pattern which might not be consistent with the task at hand. Adding thruster modules to the USR thus opens up a whole new range of application areas as the robot can achieve forward, vertical and sideways motion without performing undulatory motion.

The fact that the USRs have a floating base and links interconnected by actuated joints, poses challenges in terms of controlling the autonomous snake robot. One part of the controlling scheme is the control allocation where control commands are distributed to actuators. The control allocation problem for a USR with thrusters concerns both joint control allocation and thrust allocation. While the joint control allocation can be handled using inverse kinematics, the thrust allocation is handled using a thrust allocation algorithm. The thrust allocation of USRs is the topic of this master thesis.

1.2 Problem Description

As the USR in general is implemented with a larger number of actuators than DOFs, it can be classified as an *overactuated* system. For overactuated systems, thrust allocation can be formulated as an optimization problem where the primary objective is to minimize the error between commanded and achievable thrust. A secondary objective would be to also minimize power consumption. A common approach for thrust allocation is using Standard Inverse Matrix methods. However, the effect of using other optimization methods such as linear and quadratic programming for thrust allocation in USRs has, to the authors best knowledge, not yet been investigated. Also, the topic of constrained thrust allocation for USRs, due to saturation, rate limit or physical limitations, has not been previously addressed.

The objective of this thesis is to study thrust allocation algorithms for USRs using linear programming and quadratic programming. The scope of this thesis is:

- Perform a literature review on snake robots and control allocation methods
- Develop a Mixed Error Minimization algorithm using Linear Programming (LP)
- Develop an Error Minimization algorithm using Quadratic Programming (QP)
- Develop a Redistributed Standard Inverse (RSI) algorithm for constrained thrust allocation
- Implement methods and review performance

1.3 Contributions

The main contribution of this thesis is the implementation of several alternative thrust allocation algorithms for USRs. Algorithms for mixed error minimization using linear programming, and error minimization using quadratic programming are developed and implemented for simulation. An algorithm for constrained thrust allocation using Redistributed Standard Inverse is

also developed and implemented for simulation. The implementation and simulation is done in Matlab/SIMULINK. The performance of the algorithms is analyzed and discussed.

1.4 Thesis Outline

The thesis report is organized as follows. **Chapter 2** provides a literature review on snake robots, underwater snake robots and control allocation. In this chapter, previous work related to snake robots and control allocation is presented. **Chapter 3** gives a description of the underwater snake robot simulation model which is used for simulation. Both kinematics, kinetics and relevant parameters are presented. In **Chapter 4**, thrust allocation methods are presented, including the methods that are implemented by the author. The simulation set-up and result are presented in **Chapter 5**. Discussion of the results can also be found in this chapter. Conclusions and thoughts on further work can be found in **Chapter 6**.

Chapter 2

Literature Review

In order to get a better view on the problem at hand in this thesis, a literature study is presented in the following sections. In section 2.1, studies concerning land-based snake robots, amphibious snake robots and underwater snake robots are reviewed with a focus on modeling and kinematics. Several examples of implementation are also discussed. More extensive surveys can be found in Transeth et al. (2008) and Liljebäck et al. (2012). Section 2.2 concerns literature studies on control allocation for linear control algorithms, which includes both explicit and iterative solutions to the control allocation problem. More extensive surveys on control allocation can be found in Fossen and Johansen (2006) and Johansen and Fossen (2013).

2.1 Snake Robots

2.1.1 Biomechanical Studies of Biological Snakes

The source of inspiration for developing snake robots is largely due to the locomotion, and thus physiology, of biological snakes. Analytic studies and mathematical descriptions of snake locomotion were proposed as early as in the 1940s (Grey (1946)). The snake skeleton consists of a large amount of vertebrae (from 130 to over 400 pieces) where the range of movement for joints are about 10°-20° sideways and a few degrees up-and-down (Transeth et al. (2009)). Even though the range of movement is relatively small, the high amount of vertebrae allows the snake body to achieve a large total curvature. In addition to this, the biological snake is dependent on friction in order to generate forward motion, thus the skin surface plays an important role. The skin of snakes usually have scales which provides a smooth surface when moving forward and a coarse surface providing friction when moving backwards (Bauchot (1994)).

Research in the field of snake robots has been done for several decades. One of the pioneers within the field is Professor Shigeo Hirose at the Tokyo Institute of Technology, who developed what is considered the world's first snake robot as early as in 1972 (Hirose (1993)). One of the

well-known results of Hirose is the development of the *serpenoid curve* formulation, a mathematical description of lateral undulation. He also proposed mathematical descriptions for external factors affecting snake locomotion, such as ground friction and temperature.

Another formulation for lateral undulation, called *serpentine curve* was developed by Ma (1999). This formulation was based on mathematical modeling of the snake muscle characteristics, which was employed to derive the form of the snake body during lateral undulation motion. Ma also investigated the force required for snake locomotion. He argued that the proposed serpentine curve gave more valid results than previously proposed formulations.

Moon and Gans (1998) studied the mechanisms by which muscular activity produces curvature and propulsion. This was done by using patch electrodes to record response from the muscles of a snake body pushing against pegs. A study of the frictional properties of snake skin was performed by Hu et al. (2009). By experimentally measuring the friction coefficient of the skin, it was found that the friction coefficient on the transverse direction of the body was larger than the friction coefficient in the tangential direction.

Some of the most important movement patterns for biological snakes are described below, and are mainly based on Liljebäck et al. (2013).

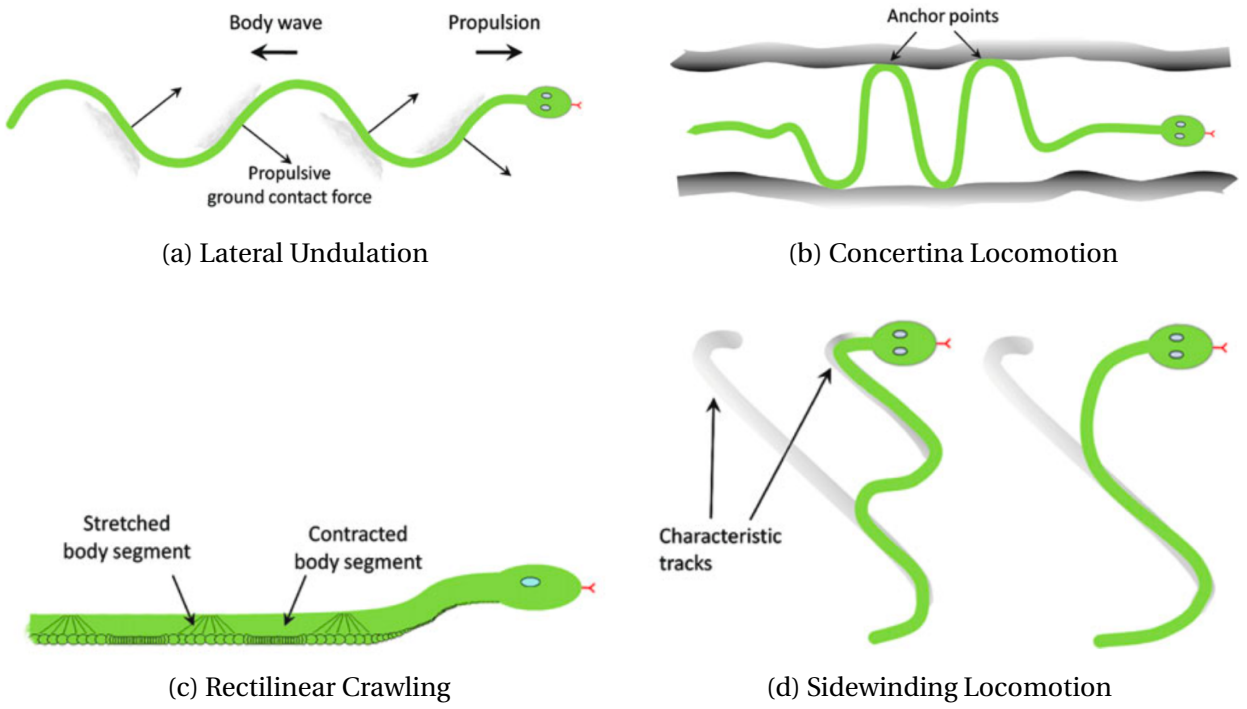
- **Lateral Undulation** (also called serpentine crawling) is maybe the most common snake locomotion form. The snake glides in a wave-like pattern where no part of the snake body is still-standing at any time. The propulsion is achieved by slightly lifting areas of the body off the ground, while generating ground contact forces at other areas of the body. The ground contact forces result from the body pushing against irregularities in the ground surface, thus using these as push-points to achieve forward motion. Lateral Undulation is illustrated in Fig. 2.1a.

- **Concertina Locomotion** is a form of locomotion often used in narrow spaces where the room for movement is small. The snake uses one part of the body as anchor points by extending the curves across the available space as shown in Fig. 2.1b, while moving the rest of the body ahead. By altering which part of the body that provides anchor points, the snake achieves forward motion.

- **Rectilinear Crawling** is a snake locomotion that allows the snake to move in a line instead of in wave-like patterns. The forward motion is achieved by using the nature of the scales as previously described in this section. The snake uses the edges of the scales to provide anchor points against the ground at some parts of the body while moving other parts of the body ahead. This is done by contracting and extending muscles at respective parts of the body. The locomotion is illustrated in Fig. 2.1c.

- **Sidewinding Locomotion** is maybe one of the more fascinating forms of snake locomotion. The mechanism is similar to concertina locomotion in that one part of the body

provides anchor points while the rest of the body moves ahead, but in this case, the surface is not the walls of a narrow space, but loose ground material such as sand. As shown in Fig. 2.1d, the snake moves in a sideways manner, leaving characteristic tracks. The snake utilizes its elongated body to "step" forward using the back part of the body as one "foot" and the front part of the body as another.



(a) Lateral Undulation

(b) Concertina Locomotion

(c) Rectilinear Crawling

(d) Sidewinding Locomotion

Figure 2.1: Biological snake locomotion forms, courtesy of Liljebäck et al. (2013)

2.1.2 Modeling and Analysis of Snake Robot Locomotion

Several of the early studies of snake robot locomotion was conducted for Flat Surface and *Sideslip Constraints*. As it was found in Hirose (1993) that the biological snake body follows the path of the head during lateral undulation, models of snake robots were developed such that the body could not move sideways (i.e. sideslip constraints). A consequence of this assumption is that nonholonomic constraints (Bloch et al. (2003)) are introduced in the equation of motion. Krishnaprasad and Tsakiris (1994) and Ostrowski and Burdick (1998) both considered the kinematics and relations between the body shape changes and the resulting body displacement for snake robots with nonholonomic constraints, as well as assessing the controllability of such mechanisms. In addition, Ostrowski and Burdick (1998) also consider the dynamics of such robots. Others who have studied the kinematics of snake robots with sideslip constraints are Matsuno and Mogi (2000) and Ma et al. (2003), while dynamics of snake robots with sideslip

constraints have been considered by Prautsch and Mita (1999), Ute and Ono (2002), Matsuno and Sato (2005), Tanaka and Matsuno (2008), Hirose (1993) and Date and Takita (2005).

Some studies have not been conducted assuming the sideslip constraints condition, but instead with the assumption that the snake robot body exhibits *anisotropic* ground friction properties, which is more similar to the biological snake. Anisotropic ground friction properties implies, as found by Hu et al. (2009), that the friction coefficients for the force tangential to the snake link body is different from the friction coefficient for the force normal to the snake link body. Models based in these properties are considered more complex and harder to analyze than those based on the sideslip constraints condition (Liljebäck et al. (2012)).

Amongst those who have studied the dynamics of snake robots with anisotropic ground friction properties is Ma (2001), who employed the Newton-Euler formulation in order to develop a two-dimensional dynamic model. In this model, both static and dynamic Coulomb ground frictions are included, and in Ma and Tadokoro (2006) the model is also extended to investigate snake locomotion on a slope. Saito et al. (2002) developed a two-dimensional model from first principles, which was used to derive properties of snake robot dynamics as well as to study controllability of joint action (Li and Shan (2008)). In Liljebäck et al. (2011) a partial feedback linearization of the model in Saito et al. (2002) is proposed in order to reduce the complexity of the model. A simplified model for snake locomotion can also be found in Liljebäck et al. (2010b), and this model is employed in Liljebäck et al. (2010a) in order to derive properties of the locomotion velocity during lateral undulation. Other two-dimensional models can be found in Kane and Levinson (2000), Grabec (2002) and Hicks (2003). Also, in Mehta et al. (2008), a 2D model including both viscous and Coulomb friction forces can be found. Models with *isotropic* Coulomb friction forces has been considered by Chernousko (2005) and Nilsson (2004).

Also three-dimensional models have been developed for snake robots without the sideslip constraints condition. Examples of this are Ma et al. (2004) who developed a model using Newton-Euler formulations including static and dynamic Coulomb ground friction forces, Liljebäck et al. (2008) who developed a model by using standard equations of motion of robotic manipulators, and Transeth et al. (2008) who modeled snake robot dynamics by using the framework of non-smooth dynamics.

2.1.3 Underwater Snake Robots

Although several solutions for underwater vehicle systems have existed for a long time, the *bio-mimetic* (biologically based) approaches to underwater locomotion have been pursued only the last decades. The increased interest is due to the several potential advantages bio-mimetic locomotion systems can provide, such as increased efficiency and agility. McIsaac and Ostrowski (1999) argued that the framework used for modeling land-based snake locomotion also can be employed for eel-like locomotion. The authors approached the modeling of eel-like robots from

a geometric perspective by using fluid drag forces to capture the effect of external forces on aquatic eels. The work was extended to consider motion planning for eel-like robots and locomotion in McIsaac and Ostrowski (2000) and McIsaac and Ostrowski (2003).

In Kanso et al. (2005), a formulation of the governing equations of motion of a system of solid bodies submerged in an ideal fluid was found in order to investigate the coupling between shape changes and fluid dynamics. The model is significantly reduced considering fluid variables in order to focus on the location of the bodies and not the fluid particles. Based on the results of the study, the authors concluded it is important to consider the hydrodynamic interaction of the links. Boyer et al. (2006) formulated a three-dimensional high level continuum model based on beam theory, which allows for computing control torques as a function of the expected internal deformation of the eel-like robot's body. This model takes into account drag forces and viscous forces, representing transversal and tangential forces on the body respectively, in addition to inertial terms.

The work of Khalil et al. (2007) presented the dynamic modeling of a three-dimensional underwater eel-like robot using recursive algorithms based on the Newton-Euler equations. Both a direct dynamic model and an inverse dynamic model are considered, and the authors argued that the algorithms developed provide a useful tool for simulation as they are easy to implement for any numbers of degrees of freedom.

An amphibious snake robot was developed by Zuo et al. (2008) in order to realize the noticeable features of a snake-like robot's serpentine locomotion in water. The dynamic model that was developed included external environmental forces such as added mass, drag, fluid acceleration and buoyancy, which are the major hydrodynamic forces. The model is simple due to an effective formulation of the generalized forces, and the final model is in a closed form. In addition, the influence of the joint angle parameters on the underwater performance of the amphibious robot are investigated, both through simulations and experiments.

Porez et al. (2014) applied a Newton-Euler modeling approach for anguilliform (or eel-like) swimming robots, focusing on the hydrodynamic forces. This model was numerically integrated in real time, and has, according to the authors, significantly superior accuracy compared to computational speed ratio.

A particularly extensive model of the kinematics and dynamics of underwater snake robots has been developed by Keladisi et al. (2014c). This model takes into account the fluid contact forces such as linear and nonlinear drag forces, current effects, added mass and fluid torque effects, as well as the hydrostatic forces, namely gravitational forces and buoyancy. The three-dimensional model is for an underwater snake robot moving in a vertical plane, but can easily be altered to simulate motion in the horizontal plane. The model is also given in closed-form, which makes it well-suited and applicable for model-based control schemes.

Several studies have been done based on the model in Keladisi et al. (2014c). In Keladisi et al.

(2014d), an integral line-of-sight guidance law is proposed, which compensates for the effects of irrotational ocean currents. Keladisi et al. (2014a) presented a control-oriented version of the model, which is less complex while still taking into account all the most essential properties of the original model. An average model of the velocity dynamics suited for stability purposes was presented in Keladisi et al. (2014b).

The question of locomotion efficiency has also been addressed in the recent years. Wiens and Nahon (2012) conducted a study of how the efficiency of undulatory propulsion is affected by system conditions such as swimming speed and energy recovery. The subjects of motion and energy efficiency, or maybe rather the balance between the two, are also addressed in Keladisi et al. (2015b), Keladisi et al. (2016a) and Keladisi et al. (2015a).

Some applications for underwater snake robots might require the robot to move while maintaining a certain configuration (e.g. when hovering, performing light intervention tasks or maneuvering in a cluttered environment). Since propulsion during lateral undulation or eel-like swimming is achieved by the oscillatory motion itself, these locomotion forms alone might not be sufficient for performing the tasks mentioned above. A solution to this is to install additional effectors on underwater snake robots.

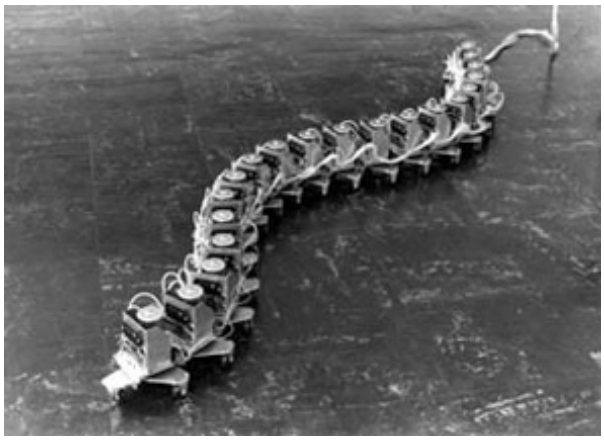
Sverdrup-Thygeson et al. (2016b) presented an underwater swimming manipulator (USM), which is a hyper-redundant AUV potentially capable of performing tasks that today traditionally are done by AUVs and ROVs. The USM is essentially a hyper-redundant underwater snake robot equipped with additional effectors such as propellers and tunnel thrusters. The authors presents models for the kinematics and dynamics of the USM as a rigid body (i.e. constant joint angles), which, in contradiction to previous works, takes into account that the links can differ in mass and length. The model is, to the authors best knowledge, the first modeling approach to take into account both hydrodynamic effects and forces produced by additional effectors.

A framework for control of the USM was presented in Sverdrup-Thygeson et al. (2016a). This control framework consists of a kinematic part which generates reference velocity signals for the joints and the USM as a rigid body, and a dynamic part which calculates generalized forces and moments and distributes these to the individual thrusters through a thrust allocation algorithm. In Keladisi et al. (2016b) the locomotion efficiency of underwater snake robots with thrusters is considered through experimental investigation.

2.1.4 Physical Implementation of Snake Robots

Over the years, many physical implementations of snake robots have been performed. When implementing models with sideslip constraints, the condition has been achieved by installing passive wheels on the snake body. An example of this is the first snake robot developed by Hirose shown in Fig. 2.2a. Other examples of robots with passive wheels are Endo et al. (1999), Togawa et al. (2000), Ma et al. (2001), Wiriyacharoensunthorn and Laowattana (2002), Mori and

Hirose (2002), Ye et al. (2004), Yu et al. (2009) and Kamegawa et al. (2009). For snake robot models without the sideslip constraint condition, the friction forces needed for propulsion are more dependent on the surface of the robot body. Some examples of implementations of snake robots without passive wheels are the PolyBot (Yim et al. (2002)), the GMD-Snake (Worst and Linnemann (1996)), the Slim Slime Snake (Ohno and Hirose (2001)) and the ACM-R7 (Ohashi et al. (2010)), which is shown in Fig. 2.2c. Other examples can be found in Nilsson (1998), Chen et al. (2007), Wright et al. (2007) and Kuwada et al. (2008).



(a) ACM III

(b) AmphiBot II

(c) ACM-R7

(d) Mamba

Figure 2.2: Examples of physical implementation of snake robots. a) shows ACM III, the first snake robot developed by Prof. Shigeo Hirose in 1972 (Hirose and Yamada (2009)), b) shows AmphiBot II, an amphibious snake robot (Crespi and Ijspeert (2006)), c) shows ACM-R7, a snake robot capable of forming a wheel (Ohashi et al. (2010)), and d) shows Mamba, a snake robot developed as an experimental platform (Liljebäck et al. (2014))

Examples of swimming robots are *REEL I* (McIsaac and Ostrowski (1999)) and *REEL II* (McIsaac and Ostrowski (2002)), the *AmphiBot* series (Crespi et al. (2005), Crespi and Ijspeert (2006) and

Porez et al. (2014)), *Perambulator III* (Li et al. (2011)), *HELIX-I* (Takayama and Hirose (2002)) and a bioinspired lamprey robot developed by Stafanini et al. (2012) as a part of the European research project "Lampetra" (Lampetra project, SSSA). One of the snake robots in the AmphiBot series, AmphiBot II, can be found in Fig. 2.2b.

The snake robot *Mamba* (Fig. 2.2d), presented by Liljebäck et al. (2014), is a modular, reconfigurable, and waterproof snake robot developed as an experimental platform to support ongoing research. The authors argued that intelligent and efficient locomotion in unknown and cluttered environments requires a robot that can *sense* and *adapt*. Thus, one of the novel contributions of the snake robot is its ability to measure environmental contact forces acting along the body by using custom-made sensors in the joints.

2.2 Linear Control Allocation

In general, control allocation is the task of distributing control commands to individual effectors in order to achieve desired forces and moments. The overactuated control allocation problem can be identified as a control allocation problem containing a larger number of effectors than DOFs to be controlled. It is usually desirable to choose control allocation model such that it is on a linear form, i.e. on the form $b = ax$ (Johansen and Fossen (2013)). One way of obtaining a linear model is by using the extended thrust formulation, e.g. Lindfors (1993), Sørdalen (1997).

For the case of *unconstrained* overactuated linear control allocation, there are virtually an infinite number of solutions solving the allocation problem. The control allocation problem can therefore be viewed as an optimization problem with the objective of finding the optimal solution amongst all the solutions. The optimal solution can be found explicitly by minimizing a least-squares cost function using *generalized inverses*, which can be derived using Lagrange multipliers (Fossen (1994), Bordignon and Durham (1995)). However, a generalized inverse might not be found if singularities occur in the problem. In such cases, regularization methods such as *damped least-squares inverse* (Berge and Fossen (1997)) and *singular value decomposition* (Sørdalen (1997)) can be applied in order to handle singularities. Both of these methods handle singularities without resorting to iterations, and therefore provides explicit solutions.

In reality, constraints have to be considered when handling control allocation problems. The *constrained* optimization problem takes into account system constraints such as physical effector limitations, actuator saturation, forbidden sectors and overload of the power system. The *redistributed pseudo-inverse method* presented in Virnig and Bodden (1994) is a constrained allocation method which decomposes the allocation problem based on which elements are saturated. This method is simple and effective, but does not guarantee a feasible solution or that the solution found is optimal. Another constrained allocation method is the *daisy chain method* (Buffington and Enns (1996)). In this method, the effectors are divided into prioritized groups

where the settings of the whole group are frozen if one or more of the effectors in the group are saturated. A modified daisy chain method has been proposed by Kim et al. (2013), addressing limitations of the original method. In Durham (1993), an explicit solution, referred to as the "direct method", was presented. This method was also considered in Bordignon and Durham (1995) where certain drawbacks of the method are addressed.

Another way of handling the constrained control allocation problem is by implementing the constraints as input constraints and minimize the error between the allocated virtual control input and the desired control input. This is done by implementing a slack variable representing the error to be minimized. By using the one-norm or the infinity-norm to formulate the cost function, the optimization problem can be identified as a *linear program* which can be solved iteratively using numerical linear programming algorithms (e.g. Paradiso (1991), Bodson (2002), Lindfors (1993), Bodson and Frost (2011)). Similarly, by using the two-norm, the optimization problem can be identified as a *quadratic program* which can be solved iteratively using numerical quadratic programming methods (e.g. Härkegård (2002), Petersen and Bodson (2006), Petersen and Bodson (2005). Common numerical methods for solving the programs are the *simplex method*, *active set methods* and *interior point methods* (Nocedal and Wright (2006)).

Chapter 3

Snake Robot Simulation Model

In order to validate the thrust allocation scheme developed in this thesis, simulations are performed using a snake robot simulation model implemented in Matlab/SIMULINK (Math Works Inc.). The following sections give a description of the kinematics and kinetics formulations used in this simulation model, in addition to relevant parameters and properties. The simulation model is developed by researchers at NTNU (Sverdrup-Thygeson et al. (2018)).

3.1 Kinematics

The modeling can be divided into kinematics and kinetics. While kinetics concerns the forces causing the motion, the kinematics describes the motion in a geometrical context by the use of so called *reference frames*. When looking at a craft in a marine environment in 6 degrees of freedom (DOF), it is convenient to use two kinematic reference systems, one describing the position of the craft in ocean space (earth-fixed coordinates) and one describing the craft motion in relation to itself (body-fixed reference frame). However, for snake robots, which are made up of several links, local reference frames for each link are also needed in order to obtain a complete description of the craft motion.

3.1.1 Reference Frames

A general description of relevant reference frames are given in this section.

NED (Earth-Fixed Reference Frame)

The *North-East-Down* coordinate system describes position $\{n\} = (x_n, y_n, z_n)$ relative to the north, east and down direction on the earth surface as defined in the World Geodetic System (1984). The origin of this reference system is fixed and thus rotates with the earth. However, as the

velocity of a marine craft usually is small, the NED frame can be approximated as an inertial coordinate system which provides an *inertial frame*.

BODY (Body-Fixed Reference Frame)

This reference frame describes motion in a coordinate system $\{b\} = (x_b, y_b, z_b)$ that is body-fixed. The origin is placed at some fixed point on the craft, usually the geometric center such that the body axis coincides with the principle axis of inertia for the craft. This reference frame is suitable for describing craft velocity in 6 DOF. Fig. 3.1 shows the general connection between the body-fixed reference frame $\{b\}$ and the 6 DOF velocities often called *sway*, *surge*, *heave*, *roll*, *pitch* and *yaw*.

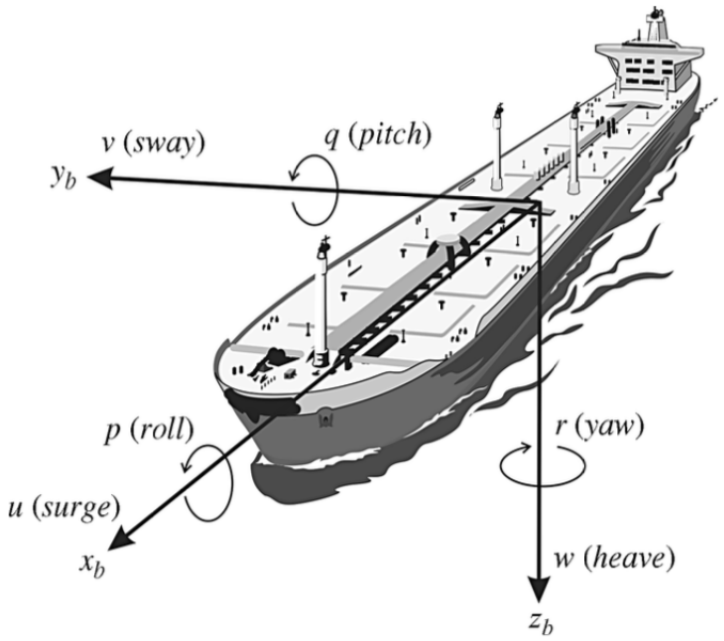


Figure 3.1: 6 DOF velocities in body-fixed reference frame {b} (Fossen (2011))

Local Reference Frames

A snake robot is made up of links interconnected by joints. This means that a snake robot with n links has $n-1$ joints. In the case of this simulation model, all joints are revolute one-parameter joints, i.e. they rotate around only one axis. The joint angle for each joint is defined as θ_i, for $i = 1 \ldots n-1$. Each link has a local reference frame which can be related to the previous link frame through the joint angle and the length of the link. Fig. 3.2 shows the local reference frames relative to each other.

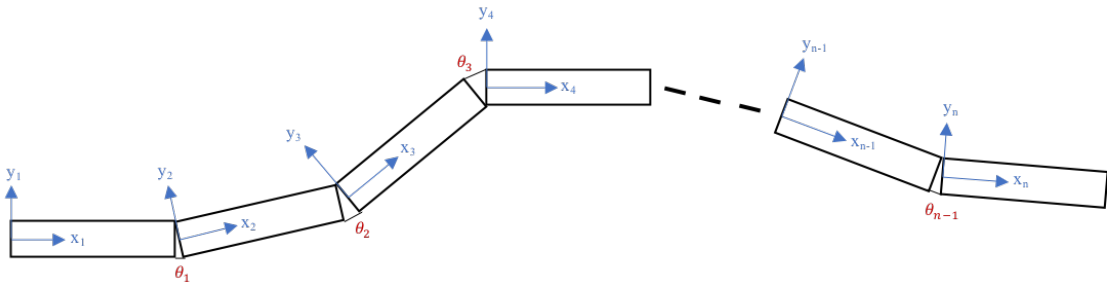


Figure 3.2: Snake robot links with local coordinate systems shown in 2D

3.1.2 Transformation Between Reference Frames

In order to represent motion in the inertial frame, transformations between reference frames are necessary. The rotation between two three-dimensional spaces are called the *special orthogonal group of order three* and is defined by

$$\boldsymbol{SO}(3) := \left\{ \boldsymbol{R} \in \mathbb{R}^{3\times 3} \middle| \det(\boldsymbol{R}) = 1, \boldsymbol{R}^{-1} = \boldsymbol{R}^T \right\} \tag{3.1}$$

The rotation matrix $\boldsymbol{R}$ represents a rotation of the three-dimensional coordinate frame around a fixed point. In addition, a *translation* of the reference frame can be added to the representation by adding a vector $\boldsymbol{t} \in \mathbb{R}^3$. Thus, a point $\boldsymbol{v}$ expressed in a certain three-dimensional reference frame can be expressed in another reference frame by the relation $\boldsymbol{v}' = \boldsymbol{R}\boldsymbol{v} + \boldsymbol{t}$ where $\boldsymbol{v}'$ represents the same point. Consequently, a *homogeneous* representation can be formulated as

$$\begin{bmatrix} \boldsymbol{v}' \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{v} \\ 1 \end{bmatrix} \tag{3.2}$$

where the middle matrix is an element of the *special Euclidean group of order three* defined as

$$\boldsymbol{SE}(3) := \left\{ \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{4\times 4} \middle| \boldsymbol{R} \in \boldsymbol{SO}(3), \boldsymbol{t} \in \mathbb{R}^3 \right\} \tag{3.3}$$

Unit Quaternions

Unit quaternions can be used to express the motion of the body-fixed reference frame relative to the inertial frame. This method of representation is used in order to avoid the possible representation singularities of the more commonly used Euler angles (Fossen (2011)). The unit quaternion $\boldsymbol{q}$ is made up of a complex number with on real part denoted η and three imaginary parts denoted $\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 & \varepsilon_2 & \varepsilon_3 \end{bmatrix}^T$, and satisfies the property $\boldsymbol{q}^T\boldsymbol{q} = 1$. A set of unit quaternions are defined as

$$\boldsymbol{Q} := \left\{ \boldsymbol{q} \middle| \boldsymbol{q}^T \boldsymbol{q} = 1, \quad \boldsymbol{q} = \begin{bmatrix} \eta & \boldsymbol{\varepsilon}^T \end{bmatrix}^T, \quad \boldsymbol{\varepsilon} \in \mathbb{R}^3 \quad \text{and} \quad \eta \in \mathbb{R} \right\} \tag{3.4}$$

The elements making up the unit quaternion are defined as

$$\begin{aligned} \eta &:= \cos\frac{\beta}{2} \\ \boldsymbol{\varepsilon} &:= \boldsymbol{\lambda} \sin\frac{\beta}{2} \end{aligned} \tag{3.5}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^3$ are unit vectors that are parallel to the axis of rotation and β is the angle NED is rotated. The unit quaternion can thus be written as

$$\boldsymbol{q} = \begin{bmatrix} \eta \\ \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix} = \begin{bmatrix} \cos\frac{\beta}{2} \\ \lambda_1 \sin\frac{\beta}{2} \\ \lambda_2 \sin\frac{\beta}{2} \\ \lambda_3 \sin\frac{\beta}{2} \end{bmatrix} \in \boldsymbol{Q}, \qquad 0 \leq \beta \leq 2\pi \tag{3.6}$$

The Cartesian body frame is defined at the first link of the snake robot, hereby called the *base frame*. The transformation between the base frame $\{b\}$ and the inertial frame $\{n\}$ can be represented by the quaternion transformation matrix defined as

$$\boldsymbol{R}_b^n(\boldsymbol{q}) := \boldsymbol{R}_{\eta,\boldsymbol{\varepsilon}} = \boldsymbol{I}_{3\times 3} + 2\eta \boldsymbol{S}(\boldsymbol{\varepsilon}) + 2\boldsymbol{S}^2(\boldsymbol{\varepsilon}) \tag{3.7}$$

where $\boldsymbol{S}(\boldsymbol{\varepsilon})$ is the skew-symmetric matrix defined as

$$\boldsymbol{S}(\boldsymbol{\varepsilon}) = -\boldsymbol{S}^T(\boldsymbol{\varepsilon}) = \begin{bmatrix} 0 & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & 0 & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & 0 \end{bmatrix}, \quad \boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix} \tag{3.8}$$

The transformation matrix allows for the body velocities to be expressed in the inertial frame. The transformation of the linear velocities can be expressed as

$$\dot{\boldsymbol{p}}^n = \boldsymbol{R}_b^n(\boldsymbol{q})\, \boldsymbol{v}^b \tag{3.9}$$

where $\boldsymbol{v}^b$ is the body-fixed velocity vector and $\dot{\boldsymbol{p}}^n$ is the inertial frame (NED) velocity vector. The transformation of the angular velocities can be expressed as

$$\dot{\boldsymbol{q}} = \boldsymbol{T}_q(\boldsymbol{q})\boldsymbol{\omega}^b \tag{3.10}$$

where $\boldsymbol{\omega}^b$ is the body-fixed angular velocity vector and $\boldsymbol{T}_q(\boldsymbol{q})$ is the transformation matrix for angular velocities given by

$$\boldsymbol{T}_q(\boldsymbol{q}) = \frac{1}{2}\begin{bmatrix} -\boldsymbol{\varepsilon}^T \\ \eta \boldsymbol{I}_{3\times 3} + \boldsymbol{S}(\boldsymbol{\varepsilon}) \end{bmatrix} \tag{3.11}$$

which is calculated using the differential equation for the transformation matrix in Eq. 3.7 (Fossen (2011)). All body-fixed motion can thus be expressed in the inertial frame by the 6 DOF kinematic equation of motion:

$$\begin{gathered} \dot{\boldsymbol{\eta}} = \boldsymbol{T}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \Updownarrow \\ \begin{bmatrix} \dot{\boldsymbol{p}}^n \\ \dot{\boldsymbol{q}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}_b^n(\boldsymbol{q}) & \boldsymbol{0}_{3\times 3} \\ \boldsymbol{0}_{4\times 3} & \boldsymbol{T}_q(\boldsymbol{q}) \end{bmatrix} \begin{bmatrix} \boldsymbol{v}^b \\ \boldsymbol{\omega}^b \end{bmatrix} \end{gathered} \tag{3.12}$$

where the vector $\boldsymbol{\eta} = \begin{bmatrix} \boldsymbol{t} & \boldsymbol{q} \end{bmatrix}^T \in \mathbb{R}^7$ describes the position and orientation of the base frame in inertial frame and the vector $\boldsymbol{\nu} \in \mathbb{R}^6$ contains all the body-fixed velocities of the base frame in 6 DOF; three translational velocities and three angular velocities. The matrix $\boldsymbol{T}(\boldsymbol{\eta})$ is the transformation matrix for all motion between base and inertial frame, which, when using unit quaternions, is a nonquadratic matrix.

Transformation between local reference frames

The position and orientation of each link can be defined in the inertial frame by

$$\boldsymbol{H}_i = \begin{bmatrix} \boldsymbol{R}_i & \boldsymbol{t}_i \\ 0 & 1 \end{bmatrix} \quad \in \boldsymbol{SE}(3), \quad i = 1 \dots n \tag{3.13}$$

The transformation between local link frame and inertial frame can thus be described using the transformation matrix

$$\boldsymbol{T}(\boldsymbol{H}_i) = \begin{bmatrix} \boldsymbol{R}_i & \boldsymbol{S}(\boldsymbol{t}_i)\boldsymbol{R}_i \\ \boldsymbol{0}_{3\times 3} & \boldsymbol{R}_i \end{bmatrix} \quad \in \mathbb{R}^{6\times 6} \tag{3.14}$$

with the inverse explicitly given as

$$\boldsymbol{T}^{-1}(\boldsymbol{H}_i) = \begin{bmatrix} \boldsymbol{R}_i^T & -\boldsymbol{R}_i^T\boldsymbol{S}(\boldsymbol{t}_i) \\ \boldsymbol{0}_{3\times 3} & \boldsymbol{R}_i^T \end{bmatrix} \tag{3.15}$$

Moreover, a mapping from one local frame of link $i+1$ to the frame of link i can be performed by utilizing that the joint angle θ_i is known. Thus, the transformation $\boldsymbol{A}_i(\theta_i)$ can be defined as

$$\begin{aligned} \boldsymbol{A}_i(\theta_i) &= \boldsymbol{A}_i(0)\begin{bmatrix} \boldsymbol{R}_{\theta_i} & \boldsymbol{0} \\ 0 & 1 \end{bmatrix} \quad \in \boldsymbol{SE}(3), \quad i = 1 \dots n-1 \\ \boldsymbol{A}_i(0) &= \begin{bmatrix} \boldsymbol{I} & l_i e_1 \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{3.16}$$

where l_i is the length of the link and $e_1 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$ as the local x-axis always is defined along the length of the link body. The matrix $\boldsymbol{R}_{\theta_i}$ is the Euler angle rotation matrix:

$$\boldsymbol{R}_{x,\theta_i} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}, \quad \boldsymbol{R}_{y,\theta_i} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}, \quad \boldsymbol{R}_{z,\theta_i} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.17}$$

where the subscript denotes about which axis θ_i rotates. This mapping can be used to express link i in the inertial frame by the recursive formula

$$\begin{aligned} \boldsymbol{H}_i &= \boldsymbol{H} \\ \boldsymbol{H}_{i+1} &= \boldsymbol{H}\boldsymbol{A}_1(\theta_1)\boldsymbol{A}_2(\theta_2)\dots\boldsymbol{A}_i(\theta_i) \end{aligned} \tag{3.18}$$

Finally, the 6 DOF body velocities of link i can be found recursively by

$$\begin{aligned} \boldsymbol{\nu}_1 &= \boldsymbol{\nu} \\ \boldsymbol{\nu}_{i+1} &= \boldsymbol{T}(\boldsymbol{H}_i)(\boldsymbol{A}_i(\theta_i))\boldsymbol{\nu}_i + \boldsymbol{a}_i\dot{\boldsymbol{\theta}}_i \end{aligned} \tag{3.19}$$

where $\boldsymbol{a}_i$ describes about which axis the angular velocity of joint i is directed. The transformation may be expressed as a Jacobian matrix:

$$\begin{aligned} \boldsymbol{J}_1 &= \begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \dots & \boldsymbol{0} \end{bmatrix} \\ \boldsymbol{J}_{i+1} &= \boldsymbol{T}(\boldsymbol{H}_i)(\boldsymbol{A}_i(\theta_i))\boldsymbol{J}_i + \begin{bmatrix} 0 & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{a}_i & \boldsymbol{0} & \dots & \boldsymbol{0} \end{bmatrix} \end{aligned} \tag{3.20}$$

which maps the velocity vector $\boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\nu}^T & \dot{\boldsymbol{\theta}}^T \end{bmatrix}^T \in \mathbb{R}^{n+5}$ containing the generalized 6 DOF velocity vector $\boldsymbol{\nu}$ in addition to the joint angle velocities $\dot{\boldsymbol{\theta}}$ to the generalized 6 DOF velocity vector $\boldsymbol{\nu}_i$ for link i such that

$$\boldsymbol{\nu}_i = \boldsymbol{J}_i\boldsymbol{\zeta} \tag{3.21}$$

3.2 Kinetics

As previously mentioned the kinetics describes the forces causing the motion of a body. These forces can be rigid body forces, hydrodynamic forces and excitation forces such as thrust and waves. The equation of motion sums up all forces acting on the craft, and thus describes the dynamics of the craft. For this simulation model, the equations of motion for the snake robot expressed in base link frame is given as

$$\begin{aligned} \dot{\boldsymbol{\eta}} &= \boldsymbol{T}(\boldsymbol{\eta})\boldsymbol{\nu} \\ \boldsymbol{M}(\boldsymbol{\theta})\dot{\boldsymbol{\zeta}} + \boldsymbol{C}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta} + \boldsymbol{D}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta} + \boldsymbol{g}(\boldsymbol{\eta},\boldsymbol{\theta}) &= \boldsymbol{B}(\boldsymbol{\theta})\boldsymbol{u} + \boldsymbol{\tau} \end{aligned} \tag{3.22}$$

The term $\boldsymbol{M}(\boldsymbol{\theta})\dot{\boldsymbol{\zeta}}$ describes the mass forces, the term $\boldsymbol{C}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta}$ describes the centripetal and coriolis effects, the term $\boldsymbol{D}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta}$ describes the drag effects and $\boldsymbol{g}(\boldsymbol{\eta},\boldsymbol{\theta})$ describes the hydrostatic forces. These terms will be more thoroughly explained in the following sections. The vector $\boldsymbol{\tau} = \begin{bmatrix} \mathbf{0}^T & \boldsymbol{\tau}_q^T \end{bmatrix}^T \in \mathbb{R}^{n+5}$ contains the joint torques applied by the motors. It can be noted that the $\boldsymbol{\tau}$-vector contains a zero vector as the mechanical forces only acts on the joints. The term $B(\boldsymbol{\theta})\boldsymbol{u}$ describes the thruster forces, and will be thoroughly explained in Chap. 4.

3.2.1 Mass Forces

The mass forces are made up of rigid body forces and hydrodynamic forces, also called *added mass*. The rigid body mass is simply the mass of the body itself, while the added mass forces are hydrodynamic pressure induced forces developing when the craft accelerates (Faltinsen (1990)). As the snake robot is made up of many links in a chain, the mass matrix contains the sum of the mass for each link transformed to the base link reference frame through the link Jacobian found in Eq. 3.20:

$$\boldsymbol{M}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \boldsymbol{J}_i^T(\boldsymbol{\theta})\boldsymbol{M}_i\boldsymbol{J}_i(\boldsymbol{\theta}), \qquad \boldsymbol{M}_i = \boldsymbol{M}_{R,i} + \boldsymbol{M}_{A,i} \tag{3.23}$$

The hydrodynamic interaction between links has been neglected in this model, which leaves the link mass matrices independent of robot configuration. As the link bodies can be modeled as cylinders, the matrices for the rigid body and added mass are given by

$$\boldsymbol{M}_{R,i} = \begin{bmatrix} m_i \boldsymbol{I} & m_i \boldsymbol{S}^T(\boldsymbol{r}_{g,i}) \\ m_i \boldsymbol{S}(\boldsymbol{r}_{g,i}) & \boldsymbol{I}_{R,i} \end{bmatrix} \tag{3.24}$$

$$\boldsymbol{M}_{A,i} = \rho \pi r^2 l_i C_a \begin{bmatrix} \alpha_i & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \frac{1}{2} l_i \\ 0 & 0 & 1 & 0 & -\frac{1}{2} l_i & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -\frac{1}{2} l_i & 0 & \frac{1}{3} l_i^2 & 0 \\ 0 & \frac{1}{2} l_i & 0 & 0 & 0 & \frac{1}{3} l_i^2 \end{bmatrix} \tag{3.25}$$

where m_i is the mass of link i, $\boldsymbol{r}_{g,i}$ is the location of the centre of gravity of link i in the local link frame, $\boldsymbol{I}_{R,i}$ is the rigid body inertia matrix of link i, ρ is the density of water, r is the radius of the cylinder representing the link body, l_i is the link length and C_a is the added mass coefficient. α_i is a parameter which allows added mass in surge to be added. It can be noted that the matrix in Eq. 3.25 has off-diagonal terms due to the fact that is formulated in a reference frame which does not give three planes of symmetry.

3.2.2 Centripetal and Coriolis Forces

The centripetal and Coriolis forces are described by

$$\begin{aligned} \boldsymbol{C}(\boldsymbol{\theta}, \boldsymbol{\zeta}) &= \sum_{i=1}^{n} \left[\boldsymbol{J}_i^T(\boldsymbol{\theta}) \boldsymbol{M}_i \dot{\boldsymbol{J}}_i(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) - \boldsymbol{J}_i^T(\boldsymbol{\theta}) \boldsymbol{W}_i(\boldsymbol{\theta}, \boldsymbol{\zeta}) \boldsymbol{J}_i(\boldsymbol{\theta}) \right] \\ \boldsymbol{W}_i(\boldsymbol{\theta}, \boldsymbol{\zeta}) &= \begin{bmatrix} 0 & \boldsymbol{S}(\{\boldsymbol{M}_i \boldsymbol{\nu}_i\}_v) \\ \boldsymbol{S}(\{\boldsymbol{M}_i \boldsymbol{\nu}_i\}_v) & \boldsymbol{S}(\{\boldsymbol{M}_i \boldsymbol{\nu}_i\}_\omega) \end{bmatrix} \end{aligned} \tag{3.26}$$

where $\{\boldsymbol{M}_i \boldsymbol{\nu}_i\}_v \in \mathbb{R}^3$ and $\{\boldsymbol{M}_i \boldsymbol{\nu}_i\}_\omega \in \mathbb{R}^3$ are the first three entries and the last three entries of $\boldsymbol{M}_i \boldsymbol{\nu}_i$ respectively, and $\dot{\boldsymbol{J}}_i$ is the time derivative of the Jacobian matrix found in Eq. 3.20. Although the term traditionally is called *the centripetal and Coriolis term*, in reality, the term takes into account forces that are more complex than the ones caused by only centripetal and Coriolis effects (Spong et al. (2006)).

3.2.3 Drag Forces

Drag forces represent the hydrodynamic damping that occurs when a craft moves relative to a fluid. The term is obtained by summing up the drag forces for the individual links transformed to the base link reference frame:

$$\boldsymbol{D}(\boldsymbol{\theta},\boldsymbol{\eta}) = \sum_{i=1}^{n} \boldsymbol{J}_i^T(\boldsymbol{\theta})\boldsymbol{D}_i(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{J}_i(\boldsymbol{\theta}), \qquad \boldsymbol{D}_i(\boldsymbol{\theta},\boldsymbol{\zeta}) = \boldsymbol{D}_{NL,i}(\boldsymbol{\theta},\boldsymbol{\zeta}) + \boldsymbol{D}_{L,i} \tag{3.27}$$

where $\boldsymbol{D}_{NL,i}(\boldsymbol{\theta},\boldsymbol{\zeta})$ is the nonlinear drag force contribution and $\boldsymbol{D}_{L,i}$ is the linear drag force contribution. The nonlinear drag forces are computed using *strip theory* (Faltinsen (1990)), where the surface integral (which is hard to compute) is replaced by a line integral over the length of the cylindrical body being partitioned into circular disk elements (McMillan et al. (1995)). The linear drag effect contribution is given by

$$\boldsymbol{D}_{L,i} = \rho\pi l_i C_{d,i} v_{ref} \begin{bmatrix} \beta_i & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & \frac{1}{2}l_i \\ 0 & 0 & 1 & 0 & -\frac{1}{2}l_i & 0 \\ 0 & 0 & 0 & \gamma_i r^2 & 0 & 0 \\ 0 & 0 & -\frac{1}{2}l_i & 0 & \frac{1}{3}l_i^2 & 0 \\ 0 & \frac{1}{2}l_i & 0 & 0 & 0 & \frac{1}{3}l_i^2 \end{bmatrix} \tag{3.28}$$

where β_i and γ_i are parameters which allow linear drag effects to be added in surge and roll. $C_{d,i}$ and v_{ref} are the drag coefficient and a reference velocity respectively. It should be noted that determining these parameters is a challenging endeavor due to their dependence on other parameters which might not be constant during craft motion.

3.2.4 Hydrostatic Forces

The hydrostatic forces are caused by the hydrostatic pressure over the craft when being in the water. They are also called *restoring forces*. As the horizontal hydrostatic forces acting on the link bodies cancel each other out, only the vertical hydrostatic forces make a contribution to this term. Also this term is given as a sum of the hydrostatic force contribution to each link transformed to the base link reference frame:

$$\boldsymbol{g}(\boldsymbol{\eta},\boldsymbol{\theta}) = \sum_{i=1}^{n} \boldsymbol{J}_i^T(\boldsymbol{\theta})\boldsymbol{g}_i(\boldsymbol{\eta},\boldsymbol{\theta}) \tag{3.29}$$

The force contribution to each link is given by

$$\boldsymbol{g}_i(\boldsymbol{\eta},\boldsymbol{\theta}) = \boldsymbol{G}_i\boldsymbol{\gamma}_i(\boldsymbol{\eta}) \tag{3.30}$$

$$\boldsymbol{G}_i = \begin{bmatrix} (\rho V_i g - m_i g)I \\ \rho V_i g \boldsymbol{S}(\boldsymbol{r}_{b,i}) - m_i g \boldsymbol{S}(\boldsymbol{r}_{g,i}) \end{bmatrix}, \qquad \in \mathbb{R}^{6\times 3} \tag{3.31}$$

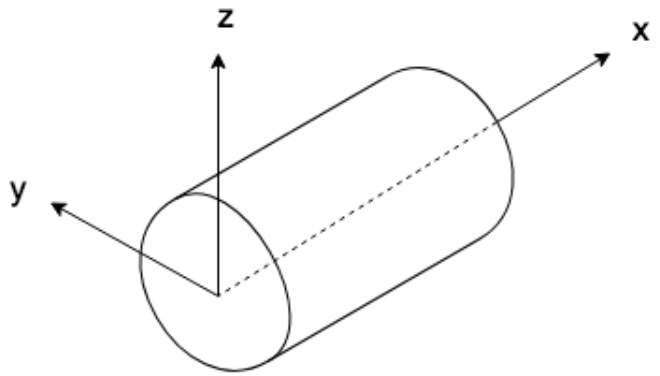


Figure 3.3: The local frame coordinate system for links

Table 3.1: Link parameters and properties

Link-type	Length [m]	Radius [m]	Number of thrusters	Direction of thrust	Placement of thrusters [x y z]
#1	0.75	0.1	2	y-direction z-direction	[0.375 0 0] [0.375 0 0]
#2	1	0.1	2	x-direction x-direction	[0.375 0.15 0] [0.375 -0.15 0]

where V_i is the volume of link i, g is the gravitational acceleration constant, $\boldsymbol{r}_{b,i}$ is the location of the centre of buoyancy of link i in the local link frame and $\boldsymbol{\gamma}_i \in \mathbb{R}^3$ is a unit vector giving the direction of gravity (which is pointing downward) for link i. The direction of gravity in inertial frame is constant and denoted $\boldsymbol{\gamma}_0$, such that

$$\boldsymbol{\gamma}_i = \boldsymbol{R}_i^T \boldsymbol{\gamma}_0 \tag{3.32}$$

3.3 Definition of Parameters and Properties

The simulation model is implemented such that link and joint properties can be chosen freely, as well as the combination of link and joint types. Two types of links are defined in the current version of the simulation model. Link-type #1 has virtual tunnel thrusters giving thrust in y- and z-direction when considering the link as a cylindrical body with the x-axis along the length of the cylinder. Link-type #2 has two thrusters giving thrust in x-direction. These thrusters are placed the outside of the link body, one on each side. The link parameters and properties can be found in Tab. 3.1. The placement of the thrusters are given in local link frame coordinates as it is shown in Fig. 3.3. It should be noted that the location of the y- and z-direction tunnel thrusters are not physically possible, but adequate for the purpose of this simulation model.

All the links are neutrally buoyant, i.e. the mass of the link is equal to the mass of displaced

water so that the hydrostatic pressure on the bottom is cancelled out by the gravity force. The links are also defined so that the centre of gravity is placed below the centre of buoyancy. This results in a naturally induced restoring moment when the link experiences roll displacement. The joints are defined as revolute joints, i.e. they provide single-axis rotation and are either x-revolute, y-revolute or z-revolute.

Chapter 4

Control Allocation

The control allocation is the phase of the control system where a control signal from a controller is allocated to the individual physical effectors on the vehicle. In general, effectors can be mechanical devices producing forces and moment such as rudders, fins, propellers and thrusters. Actuators are electromechanical effectors such as propellers and thrusters. The controller outputs a virtual control vector $\boldsymbol{\tau}_c \in \mathbb{R}^m$ using a high level motion control algorithm. This vector usually represents a number of forces and moments, m, equal to the number of degrees of freedom (DOFs) in the system.

In the case of an underwater snake robot, the input to the Dynamic Controller, which contains the high level motion control algorithm, is a set of tasks calculated using Inverse Kinematics. These tasks represent the snake robot joint angles required in order to achieve a certain desired end position and/or geometric configuration. The output from the Dynamic Controller is a vector containing commanded thrust in 6 DOF. These are subsequently allocated to the individual underwater snake robot thrusters by the Thrust Allocation algorithm. A system diagram for the snake robot control system can be found in Fig. 4.1. The contribution in this thesis concerns the content of the "Thrust allocation" block in the diagram.

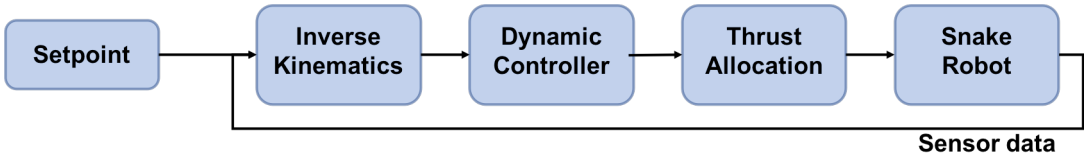


Figure 4.1: System diagram for snake robot control system

In order to calculate the thrust contribution from each thruster to the individual DOFs of the vehicle, a thrust configuration matrix, denoted $\boldsymbol{B}$, is required. This matrix maps the individual thrusters contribution based on the geometric placement of the thrusters. This matrix might be dependant on a variable $\boldsymbol{B}(\boldsymbol{\alpha})$ where $\boldsymbol{\alpha}$ is the thruster orientation vector. This is for example the case for a vehicle with azimuth thrusters. In the case of the snake robot, the thrusters are

assumed to be placed rigidly on the links making up the snake robot. This means that the thrust configuration matrix is static in each time step, but is recalculated from one time step to the next. This is because the directions of thrust for the rigidly placed thrusters are dependant on the joint angles defining the snake robot configuration.

The objective of the control allocation algorithm is to distribute forces and moments such that the desired end position and/or configuration is achieved. The algorithm is designed to map the virtual (desired) control input $\boldsymbol{\tau}_c$ into effector forces and moments such that the total of these are equal to the total of $\boldsymbol{\tau}_c$. The number of effector forces and moments are denoted p. The effector model is often modeled to be linear such that

$$\boldsymbol{\tau} = \boldsymbol{h}(\boldsymbol{u}, \boldsymbol{x}, t) = \boldsymbol{B}(\boldsymbol{x}, t)\boldsymbol{u} \tag{4.1}$$

where t is time, $\boldsymbol{x} \in \mathbb{R}^n$ is the state vector, $\boldsymbol{u} \in \mathbb{U} \subset \mathbb{R}^p$ is the control output to the effectors and $\boldsymbol{\tau} \in \mathbb{R}^m$ is the virtual effector model input which should fulfill $\boldsymbol{\tau} = \boldsymbol{\tau}_c$ in the ideal case. The subset $\mathbb{U}$ represents the set of possible values of $\boldsymbol{u}$, which is limited due to for example thruster saturation and other physical constraints. For an overactuated system, $p > m$, there might not be a unique solution to the inverse problem of calculating $\boldsymbol{u}$ given $\boldsymbol{\tau} = \boldsymbol{\tau}_c$. The main objective of the control allocation algorithm is therefore to find the optimal solution for $\boldsymbol{u}$ given a certain set of constraints.

4.1 Explicit Methods

One way of solving the control allocation problem is to calculate the inverse of the thrust configuration matrix $\boldsymbol{B}$. For an overactuated system, the $\boldsymbol{B}$-matrix will not be square, i.e. it will have more columns than rows, which means it can be assumed that it has a non-trivial null space. This, in turn, means that there is an infinite number of vectors $\boldsymbol{u}$ that satisfy Eq. 4.1 for any given $\boldsymbol{\tau}$. A common way of handling this problem is to use the generalized inverse (or pseudo-inverse) in order to calculate a unique $\boldsymbol{u}$. The much used Moore-Penrose pseudo-inverse (Golub and Loan (1983)) can be used to obtain the solution $\boldsymbol{u} = \boldsymbol{B}^\dagger \boldsymbol{\tau}_c$, and is defined as

$$\boldsymbol{B}^\dagger = \boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{B}^T)^{-1} \tag{4.2}$$

The Moore-Penrose pseudoinverse is used in several algorithms for finding the optimal control allocation. Some of these are explained below.

4.1.1 Standard Damped Inverse

Usually, a system is designed such that desirable forces and moments are possible to generate. However, in special cases, such as singularities or actuator failure, the $\boldsymbol{B}$-matrix can become

rank deficient. In such cases, the control algorithm should be able to handle the situation with reallocating thrust to available thrusters. One way of doing this is to use the standard damped inverse (or damped least-squares inverse). This method finds a solution to the equation

$$\boldsymbol{u} = \boldsymbol{C}_\varepsilon \boldsymbol{\tau}_c \tag{4.3}$$

where

$$\boldsymbol{C}_\varepsilon = \boldsymbol{B}^T(\boldsymbol{B}\boldsymbol{B}^T + \varepsilon \boldsymbol{I})^{-1} \tag{4.4}$$

and $\varepsilon \geq 0$ is a small damping constant which acts as a regularization parameter handling singular thruster configurations. When implementing this algorithm using software computation tools, it is also usual to include in the algorithm a segment making sure that entries in the inverse $\boldsymbol{B}$-matrix with an absolute value smaller than a certain boundary limit γ, is set to zero. This is to prevent the computation of negligible thrust contribution to unnecessarily increase computation time.

4.1.2 Prioritized Inverse

The prioritized inverse method largely resembles the standard damped inverse method in that it contains the Moore-Penrose pseudo-inverse with a damping constant acting as a regularization parameter. However, the method enables prioritizing of DOFs by splitting the thrust allocation problem into subproblems. A typical example of a situation where prioritized inverse is used, is for dynamic positioning of surface vessels where the heading control is prioritized over surge and sway control. For a six DOF vehicle, the linear motions can be prioritized by splitting the $\boldsymbol{B}$ into $\boldsymbol{B}_l \in \mathbb{R}^{r\times p}$ and $\boldsymbol{B}_a \in \mathbb{R}^{s\times p}$ where the subscript l denotes linear motion, the subscript a denotes angular motion, r is the number of linear DOFs and s is the number of angular DOFs. This way, the thruster outputs can be calculated by

$$\boldsymbol{u} = \boldsymbol{C}_l \boldsymbol{\tau}_l + (\boldsymbol{I}_{p\times p} - \boldsymbol{C}_l \boldsymbol{B}_l)\boldsymbol{C}_a \boldsymbol{\tau}_a \tag{4.5}$$

where

$$\begin{aligned} \boldsymbol{C}_l &= \boldsymbol{B}_l^T(\boldsymbol{B}_l\boldsymbol{B}_l^T + \varepsilon \boldsymbol{I}\boldsymbol{W}_l)^{-1} \\ \boldsymbol{C}_a &= \boldsymbol{B}_a^T(\boldsymbol{B}_a\boldsymbol{B}_a^T + \varepsilon \boldsymbol{I}\boldsymbol{W}_a)^{-1} \end{aligned} \tag{4.6}$$

are the weighted pseudo-inverse of each subproblem and $\boldsymbol{W}_l$ and $\boldsymbol{W}_a$ are diagonal matrices with weights for linear motions and angular motions respectively.

4.1.3 Inverse with Single Value Decomposition (SVD)

Another method for handling singularities in the inverse $\boldsymbol{B}$-matrix is the Singular Value Decomposition (SVD). This method performs a factorization of a real matrix (in this case the $\boldsymbol{B}$-matrix such that

$$\boldsymbol{B}_{(m\times p)} = \boldsymbol{U}_{(m\times m)}\boldsymbol{\Sigma}_{(m\times p)}\boldsymbol{V}^T_{(p\times p)} \tag{4.7}$$

where $\boldsymbol{U}$ and $\boldsymbol{V}$ are unitary matrices such that $\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}_m$ and $\boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}_p$. The matrix $\boldsymbol{\Sigma}$ is a diagonal matrix containing the singular values σ_i of the matrix $\boldsymbol{B}$. The inversion of the $\boldsymbol{B}$-matrix is obtained by

$$\boldsymbol{B}_{inv} = \boldsymbol{V}\boldsymbol{\Sigma}_{inv}\boldsymbol{U}^T \tag{4.8}$$

where

$$\boldsymbol{\Sigma}_{inv} = \begin{bmatrix} \frac{1}{\sigma_1} & 0 & \dots & & 0 \\ 0 & \frac{1}{\sigma_2} & & & \\ & & \ddots & & \\ \vdots & & & \frac{1}{\sigma_{p^\star}} & \vdots \\ & & & & 0 \\ & & & & & \ddots \\ 0 & & \dots & & 0 \end{bmatrix} \tag{4.9}$$

Here, the singular values larger than a certain threshold ($p^\star \leq p$) are inverted and gathered in the upper left corner of the diagonal matrix while the singular values smaller than the threshold are set to zero and gathered in the lower right corner. This way, the singularities in the system are taken care of, and the resulting inverted $\boldsymbol{B}$-matrix can be used for control allocation.

4.1.4 Redistributed Pseudo-Inverse (RPI)

Constrained control allocation allows for additional problem conditions to be included, namely constraints due to thruster saturation and physical limitations. An explicit method for solving control allocation problems subjected to saturation limit constraints is the redistributed pseudo-inverse method, which can be seen as an extension of the standard damped inverse method. The approach is as follows: First, solution to Eq. 4.3 is found using the damped inverse in Eq. 4.4. If any of the calculated thrust forces are above the saturation limit, the thrust vector elements are set *equal* to the saturation limit. The thrust vector is then reduced to exclude the saturated element. The thrust configuration matrix is also reduced, such that the columns cor-

responding to the saturated thrust vector element is removed. The process of solving Eq. 4.3 is then repeated. This is done until no more elements reach the saturation limit. The thrust vector elements are then reassembled, and the resulting thrust vector is the final solution.

4.2 Iterative Methods

In an optimization problem, constraints can be included such that the algorithm searches for the optimal solution within available solutions. This search can be done by using iterative methods. The objective is the same as above, i.e. to find a control output $\boldsymbol{u} \in \mathbb{U}$ which fulfills $\boldsymbol{\tau} = \boldsymbol{\tau}_c$. Given the constraints, this equality condition might not have a feasible solution. Therefore, a slack variable $\boldsymbol{s}$ is introduced. This variable serves as an error variable such that

$$\boldsymbol{B}\boldsymbol{u} = \boldsymbol{\tau}_c + \boldsymbol{s} \tag{4.10}$$

By using error minimization as the objective for the optimization algorithm, a cost function can be formulated as

$$\boldsymbol{J}_s = ||\boldsymbol{B}\boldsymbol{u} - \boldsymbol{\tau}_c|| \tag{4.11}$$

where $\boldsymbol{u}_{min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{max}$. Alternatively, a secondary objective, such as control minimization, can be included in the problem. This allows for the algorithm to find a unique optimal solution if several solutions for the primary objective exist. In this case, the cost function would be

$$\boldsymbol{J}_s = \boldsymbol{q}||\boldsymbol{B}\boldsymbol{u} - \boldsymbol{\tau}_c|| + \boldsymbol{w}||\boldsymbol{u} - \boldsymbol{u}_p|| \tag{4.12}$$

where $\boldsymbol{u}_{min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{max}$, $\boldsymbol{w}$ and $\boldsymbol{q}$ are weight vectors and $\boldsymbol{u}_p$ is a vector with some preferred thrust values. Here, the first term is dedicated to satisfy the primary objective of minimizing error, while the second term is dedicated to satisfy the secondary objective of minimizing control, i.e. the usage of thrust and thus also the power consumption. By setting the weight value $\boldsymbol{w}$ small compared to $\boldsymbol{q}$, the algorithm greatly prioritizes the error minimization over the control minimization.

4.2.1 Mixed Error Minimization using Linear Programming

The mixed error minimization algorithm is an optimization algorithm minimizing both error and control. Based on the cost function in Eq. 4.12 the optimization problem can be formulated as

$$\begin{aligned} \min_{\boldsymbol{u}\in\mathbb{R}^p, \boldsymbol{s}\in\mathbb{R}^m} \quad & \boldsymbol{q}||\boldsymbol{B}\boldsymbol{u}-\boldsymbol{\tau}_c|| + \boldsymbol{w}||\boldsymbol{u}-\boldsymbol{u}_p|| \\ \text{subject to} \quad & \boldsymbol{\tau}_c - \boldsymbol{B}\boldsymbol{u} = \boldsymbol{s}, \quad \boldsymbol{u}\in\mathbb{U} \end{aligned} \tag{4.13}$$

One way of solving this optimization problem is by using Linear Programming. This prerequisites that the norm used in the problem is the l_1-norm. In order to perform the optimization, auxiliary variables are defined to make sure the parameters to be optimized are always positive. The auxiliary variables are defined as (Johansen and Fossen (2013))

$$\begin{aligned} s_i^+ &= \begin{cases} s_i, & s_i \geq 0 \\ 0, & s_i \leq 0 \end{cases} & u_i^+ &= \begin{cases} u_i, & u_i \geq 0 \\ 0, & s_i \leq 0 \end{cases} \\ s_i^- &= \begin{cases} -s_i, & s_i \leq 0 \\ 0, & s_i \geq 0 \end{cases} & u_i^- &= \begin{cases} -u_i, & u_i \leq 0 \\ 0, & s_i \geq 0 \end{cases} \end{aligned} \tag{4.14}$$

where

$$u_i = u_i^+ - u_i^- + u_{p,i} \qquad s_i = s_i^+ - s_i^- \tag{4.15}$$

The upper and lower bound for the variables can thus be defined by

$$\begin{aligned} \boldsymbol{0} \leq \boldsymbol{u}^+ \leq \boldsymbol{u}_{max} - \boldsymbol{u}_p \qquad & \boldsymbol{0} \leq \boldsymbol{s}^+ \leq \boldsymbol{s}_{max} \\ \boldsymbol{0} \leq \boldsymbol{u}^- \leq \boldsymbol{u}_p - \boldsymbol{u}_{max} \qquad & \boldsymbol{0} \leq \boldsymbol{s}^- \leq \boldsymbol{s}_{max} \end{aligned} \tag{4.16}$$

where $\boldsymbol{u}_{max}$ is determined by the capacity of the individual thrusters and $\boldsymbol{s}_{max}$ is some boundary constraint set to prohibit the slack variable from becoming to large. This way, the mixed error optimization problem can be formulated in the form

$$\min_{\boldsymbol{x}} \boldsymbol{f}^T\boldsymbol{x} \qquad \text{subject to} \quad \begin{cases} \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq} \\ \boldsymbol{b}_l \leq \boldsymbol{x} \leq \boldsymbol{b}_u \end{cases} \tag{4.17}$$

where the matrices and vectors are defined as

$$\begin{aligned} \boldsymbol{f}^T &= \begin{bmatrix} \boldsymbol{q} & \boldsymbol{q} & \boldsymbol{w} & \boldsymbol{w} \end{bmatrix} & \boldsymbol{A}_{eq} &= \begin{bmatrix} \boldsymbol{I} & -\boldsymbol{I} & -\boldsymbol{B} & \boldsymbol{B} \end{bmatrix} \\ \boldsymbol{x} &= \begin{bmatrix} \boldsymbol{s}^+ & \boldsymbol{s}^- & \boldsymbol{u}^+ & \boldsymbol{u}^- \end{bmatrix}^T & \boldsymbol{b}_{eq} &= \boldsymbol{B}\boldsymbol{u}_p - \boldsymbol{\tau}_c \end{aligned} \tag{4.18}$$

$$\boldsymbol{b}_l = \begin{bmatrix} \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}^T \qquad \boldsymbol{b}_u = \begin{bmatrix} \boldsymbol{s}_{max} & \boldsymbol{s}_{max} & \boldsymbol{u}_{max} - \boldsymbol{u}_p & \boldsymbol{u}_p - \boldsymbol{u}_{min} \end{bmatrix}^T \tag{4.19}$$

The expression $\boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq}$ represents the equality constraints, and as seen by inspection of

Eq. 4.18, the equality constraints corresponds to the objective $\boldsymbol{s} - \boldsymbol{B}\boldsymbol{u} = \boldsymbol{B}\boldsymbol{u}_p - \boldsymbol{\tau}_c \Leftrightarrow \boldsymbol{B}(\boldsymbol{u}_p + \boldsymbol{u}) = \boldsymbol{\tau}_c + \boldsymbol{s}$. The vector $\boldsymbol{x}$ is the parameters to be optimized, and $\boldsymbol{b}_l$ and $\boldsymbol{b}_u$ are, respectively, the lower and upper boundary constraints for these parameters.

4.2.2 Error Minimization using Quadratic Programming

Another way of solving the optimization problem is to use the Quadratic Programming method. This is a way of error minimization using the squared of the l_2-norm. The resulting cost function thus becomes

$$\boldsymbol{J}_s = ||\boldsymbol{B}\boldsymbol{u} - \boldsymbol{\tau}_c||_2^2 \tag{4.20}$$

where $\boldsymbol{u}_{min} \leq \boldsymbol{u} \leq \boldsymbol{u}_{max}$. This leads to the following control allocation formulation:

$$\begin{aligned} \min_{\boldsymbol{s},\boldsymbol{u}} \quad & \left(\sum_{i=1}^{m} q_i s_i^2 + \sum_{j=1}^{p} w_j u_j^2 \right) \\ \text{subject to} \quad & \boldsymbol{B}\boldsymbol{u} = \boldsymbol{\tau}_c + \boldsymbol{s}, \quad \boldsymbol{u} \in \mathbb{U} \end{aligned} \tag{4.21}$$

Since the squared of the norm is used in the quadratic programming method, the parameters to be optimized are always positive as long as the weights $\boldsymbol{q}$ and $\boldsymbol{w}$ are positive. This represents an advantage when using the quadratic programming method compared to the linear programming method as it follows that the quadratic programming problem is strictly convex, and no auxiliary variables are needed. The problem can be transformed into a standard quadratic programming form

$$\min_{\boldsymbol{x}} \quad \frac{1}{2}\boldsymbol{x}^T \boldsymbol{H}\boldsymbol{x} \qquad \text{subject to} \quad \begin{cases} \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ \boldsymbol{A}_{eq}\boldsymbol{x} = \boldsymbol{b}_{eq} \end{cases} \tag{4.22}$$

where the matrices and vectors are defined as (Johansen and Fossen (2013))

$$\begin{aligned} \boldsymbol{H} &= 2\text{diag}(w_1, ..., w_p, q_1, ..., q_m) & \boldsymbol{A}_{eq} &= \begin{bmatrix} \boldsymbol{B} & -\boldsymbol{I} \end{bmatrix} \\ \boldsymbol{x} &= \begin{bmatrix} \boldsymbol{u} & \boldsymbol{s} \end{bmatrix}^T & \boldsymbol{b}_{eq} &= \boldsymbol{\tau}_c \\ \boldsymbol{A} &= \begin{bmatrix} -\boldsymbol{I} & \boldsymbol{0} \\ \boldsymbol{I} & \boldsymbol{0} \end{bmatrix} & \boldsymbol{b} &= \begin{bmatrix} -\boldsymbol{u}_{min} & \boldsymbol{u}_{max} \end{bmatrix}^T \end{aligned} \tag{4.23}$$

Here, the boundary constraints for $\boldsymbol{u}$ are implemented as inequality constraints. It can also be noted that the slack variable $\boldsymbol{s}$ is not constrained. Constraining $\boldsymbol{s}$ is generally optional, and

can be implemented in cases where it is thought to improve the functionality of the algorithm. It can also be noted that this quadratic programming algorithm will always find a feasible and unique solution due to the lack of auxiliary variables. A secondary objective is therefore not needed, as opposed to in the linear programming method.

Chapter 5

Simulation and Results

Two optimization algorithms, mixed error minimization using linear programming (LP) and error minimization using quadratic programming (QP), are implemented in Matlab/SIMULINK based on the methods described in section 4.2.1 and section 4.2.2 respectively. The LP and QP source code can be found in App. B.1 and App. B.2 respectively. An algorithm for constrained thrust allocation using redistributed pseudo-inverse (RPI) is also implemented based on the method described in section 4.1.4. The RPI source code can be found in App. B.3. Simulations are performed using the existing underwater snake robot simulation model described in Chap. 3. In order to analyze the performance of the developed algorithms, two simulation cases are defined. In the first case, the thrust allocation is considered unconstrained, i.e. the thrusters have virtually unlimited capacity. In the second case, the thrust allocation is constrained and a saturation limit for the thrusters is implemented. In both cases, simulations are performed for both planar (2D) and three-dimensional (3D) motion. In the first case, simulations are also performed using a pre-implemented standard damped inverse (SDI) algorithm for comparison. This algorithm is based on the method described in section 4.1.1.

5.1 Simulation Set-Up

All simulations are performed using the same configuration for the snake robot model. The snake robot is made up of 5 links interconnected by four z-revolute joints. The first two and the last two links are Link-type #1, while the middle link is Link-type #2. The link types are described in section 3.3 and their properties can be found in Tab. 3.1. The simulations are performed using a fixed time-step of 0.001 and an *ode3* solver. The input to the simulation model is desired positions for the end effector (i.e. the end of the last link), which changes at certain points in the simulation time course. In both cases, two simulations are performed. The first simulates planar motion, i.e. the z-coordinate for the desired positions is kept constant equal to zero. The second simulation simulates three-dimensional motion. The desired positions for the simulations are

listed in Tab. 5.1 and Tab. 5.2 respectively. In both simulations, the first position is the initial position of the end effector i.e. the end of link 5, expressed in the base link reference frame.

Table 5.1: Desired positions for simulation of planar motion

Time (t)	$0 \leqslant t < 5$	$5 \leqslant t < 20$	$20 \leqslant t < 40$	$40 \leqslant t \leqslant 70$
Position $[xyz]$	[4 0 0]	[5 3 0]	[7 3 0]	[7 4 0]

Table 5.2: Desired positions for simulation of three-dimensional motion

Time (t)	$0 \leqslant t < 5$	$5 \leqslant t < 20$	$20 \leqslant t < 40$	$40 \leqslant t \leqslant 70$
Position $[xyz]$	[4 0 0]	[4 2 1]	[4 2 2]	[5 3 2]

The simulation parameters used in the thrust allocation algorithms can be found in Tab. 5.3. In the STI/RPI algorithm, ε denotes the damping coefficient in the standard inverse matrix and α denotes the boundary limit for which entries in the standard inverse matrix are set to zero. In the LP algorithm, $\boldsymbol{u}_p$, which is the preferred thrust values, is set to zero for all thrusters. This represents that it is preferable to use as little thrust as possible in order to minimize power consumption. $\boldsymbol{s}_{max}$ denotes the maximum slack variable value. The slack variable represents the error between the generalized commanded and actual thrust. The parameter is set to the 1-norm of the commanded thrust vector, i.e. the total magnitude of the commanded thrust. In both the LP and QP algorithms, the weight coefficient w is set to a large value compared to the weight coefficient q. This reflects the fact that the primary objective of both algorithms is to minimize the slack variable, i.e. the error between commanded and actual thrust.

Table 5.3: Thrust allocation algorithm parameters

STI/RPI		**LP**		**QP**	
ε	$= 0.02$	$\boldsymbol{s}_{max}$	$= \|\|\boldsymbol{\tau}_c\|\|_1$	w	$= 50$
γ	$= 10^{-15}$	$\boldsymbol{u}_p$	$= \mathbf{0}$	q	$= 1$
		w	$= 1$		
		q	$= 10^{-5}$		

5.2 CASE 1: Unconstrained Thrust Allocation

In CASE 1, the thrust allocation is considered unconstrained. This is done by setting a virtual constraint on the individual thruster forces u_i which is high enough that the saturation limit

should never be reached. In the simulations performed in CASE 1, the constraints are set to $u_{max,i} = 1000$ N and $u_{min,i} = -1000$ N. Simulations are performed for both planar and three-dimensional motion using standard damped inverse, linear programming and quadratic programming algorithms for thrust allocation.

5.2.1 Results

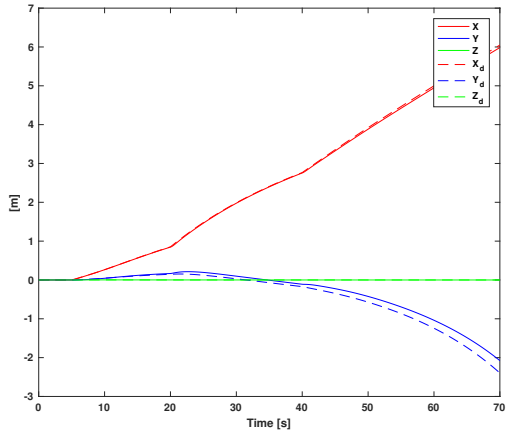
The simulation results for planar motion can be found in Fig. 5.2-5.5, as well as Fig. 5.1a, 5.1c and 5.1e. The simulation results for three-dimensional motion can be found in Fig. 5.8-5.11, as well as Fig. 5.1b, 5.1d and 5.1f. Some simulation results are summed up in Tab. 5.4 and Tab. 5.5. Additional simulation results can be found App. A.1.

Table 5.4: The absolute value of the largest errors from simulations in CASE 1

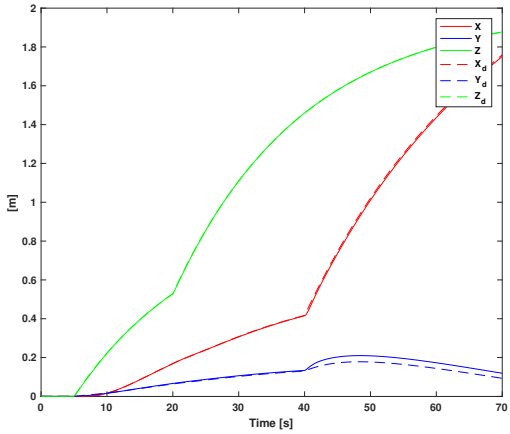
Motion	**Alloc. method**	**Surge [N]**	**Sway [N]**	**Heave [N]**	**Roll [Nm]**	**Pitch [Nm]**	**Yaw [Nm]**
2D	SDI	0.8675	0.6238	0.1110	68.4428	0.2614	0.2279
	LP	0	0	0	24.0650	0	0
	QP	0.8675	0.6238	0.1110	68.4428	0.2614	0.2279
3D	SDI	0.6158	0.5736	2.6861	46.4057	2.4024	0.1978
	LP	0	0	0	16.1217	0	0
	QP	0.6158	0.5736	2.6861	46.4057	2.4024	0.1978

Table 5.5: Maximum and minimum values for thruster forces from simulations in CASE 1

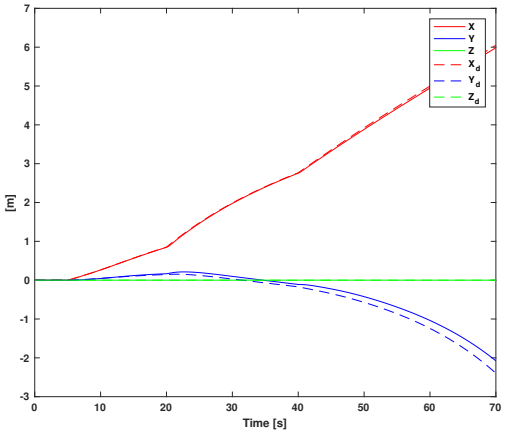
Motion	**Alloc. method**	**Value [N]**	**Thr. #1**	**Thr. #2**	**Thr. #3**	**Thr. #4**	**Thr. #5**	**Thr. #6**	**Thr. #7**	**Thr. #8**	**Thr. #9**	**Thr. #10**
2D	SDI	$u_{a,max}$	26.9167	0.6887	5.1852	2.0722	49.3103	45.9345	4.2088	6.6902	9.2462	7.7377
		$u_{a,min}$	-8.2498	-1.0730	-7.0782	-2.8405	-1.8843	-1.7385	-12.8116	-10.2962	-2.3611	-4.6535
	LP	$u_{a,max}$	23.6135	538.3219	10.0265	0.6632	98.7889	56.8220	11.1153	1000.0	8.6217	2.0981
		$u_{a,min}$	-8.5936	-1.6972	-11.0958	-1000.0	-2.4601	0	-4.1574	-2.4408	-4.4299	-699.9495
	QP	$u_{a,max}$	26.9167	0.6887	5.1852	2.0722	49.3103	45.9345	4.2088	6.6902	9.2462	7.7377
		$u_{a,min}$	-8.2498	-1.0730	-7.0782	-2.8405	-1.8843	-1.7385	-12.8116	-10.2962	-2.3611	-4.6535
3D	SDI	$u_{a,max}$	24.9688	89.2612	5.2620	56.7505	37.6405	34.6731	2.7389	27.6293	5.0224	7.2691
		$u_{a,min}$	-5.5032	-0.6459	-5.2069	-2.6290	-1.2879	-1.1649	-9.9935	-3.3279	-0.2436	-41.9085
	LP	$u_{a,max}$	24.2739	640.5721	2.5047	12.8079	73.9194	19.1996	7.1975	1000	6.1827	4.8121
		$u_{a,min}$	-7.4410	-1.0885	-4.9873	-1000	-2.3753	-0.2256	-2.2325	-15.9331	-0.5283	-742.3169
	QP	$u_{a,max}$	24.9688	89.2612	5.2620	56.7505	37.6405	34.6731	2.7389	27.6293	5.0224	7.2691
		$u_{a,min}$	-5.5032	-0.6459	-5.2069	-2.6290	-1.2879	-1.1649	-9.9935	-3.3279	-0.2436	-41.9085



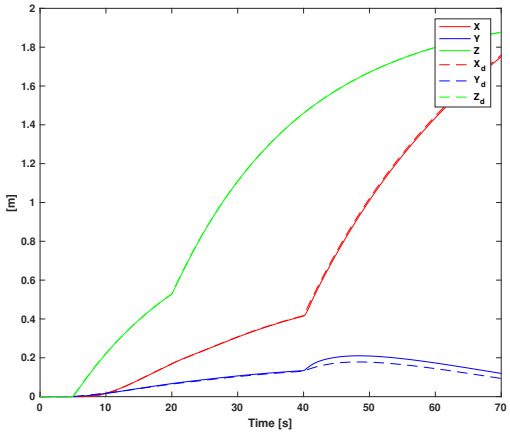
(a) 2D motion using SDI



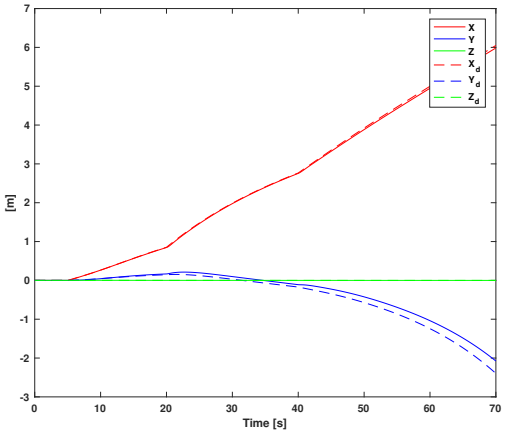
(b) 3D motion using SDI



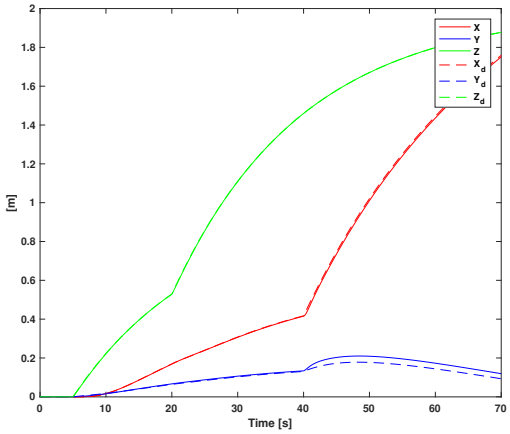
(c) 2D motion using LP



(d) 3D motion using LP



(e) 2D motion using QP



(f) 3D motion using QP

Figure 5.1: Position and desired position (dashed line) for end effector during CASE 1 simulations

Planar Motion - Standard Damped Inverse

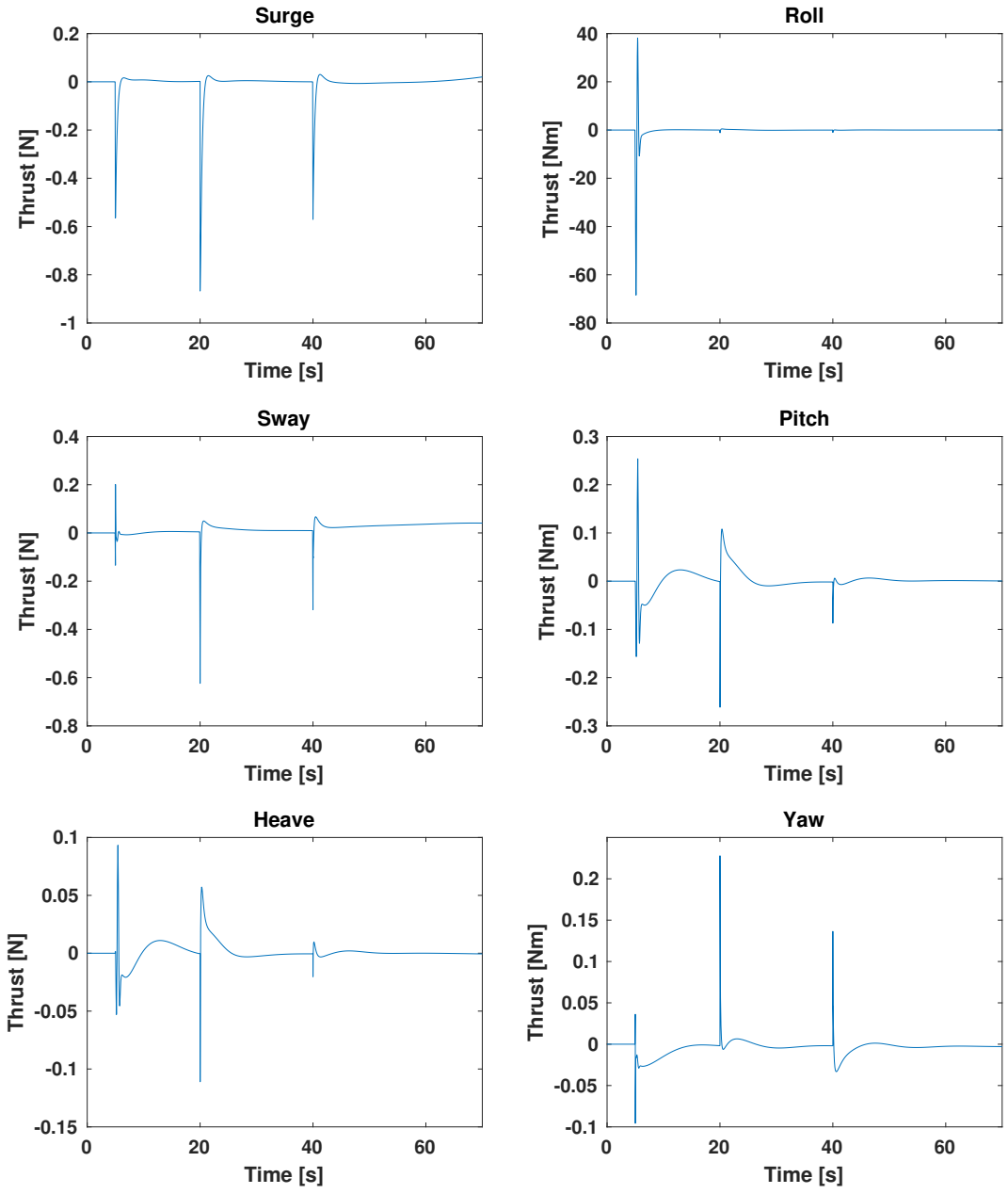


Figure 5.2: Error between commanded and actual thrust for unconstrained thrust allocation for planar motion using standard damped inverse

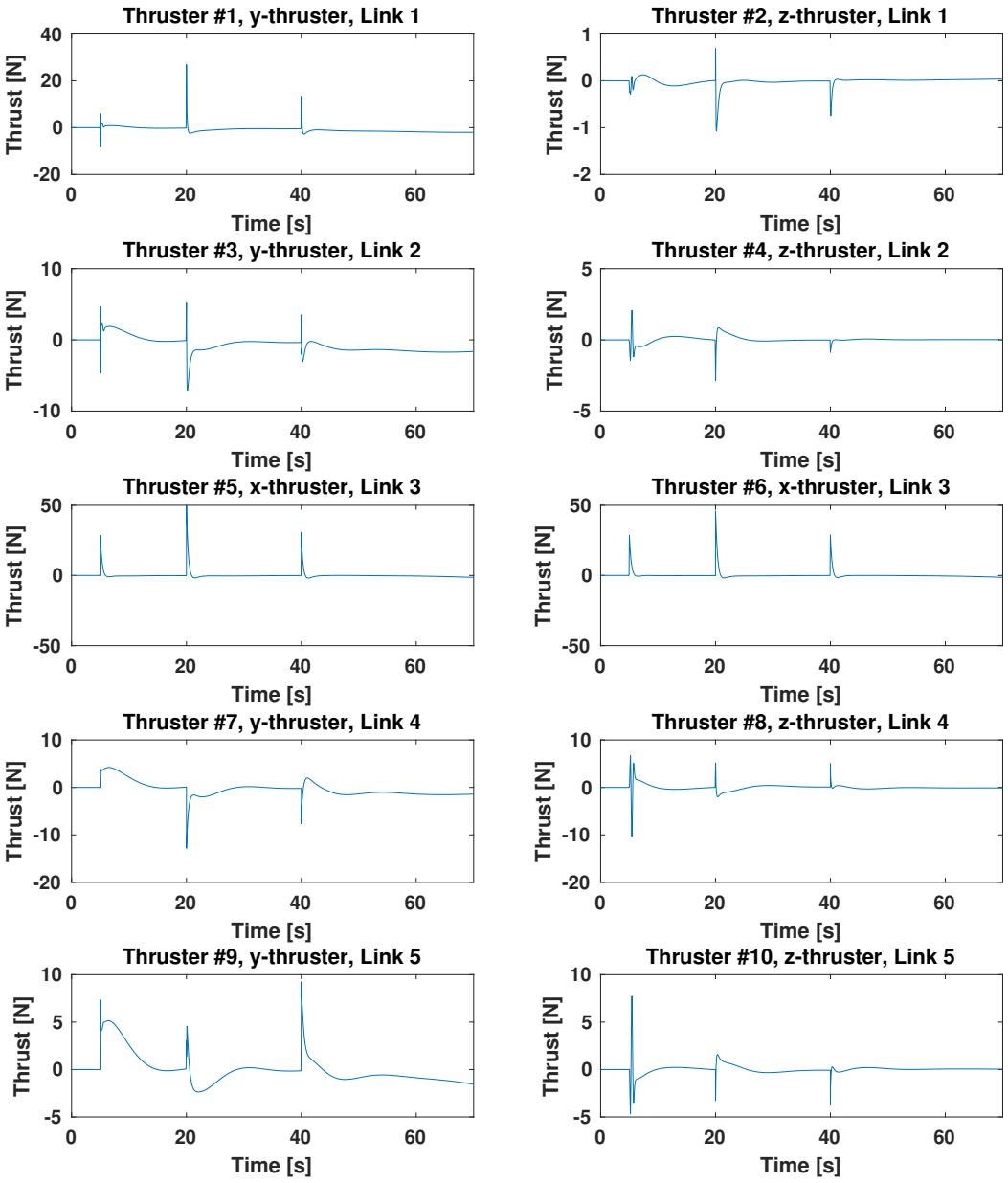


Figure 5.3: Individual thruster forces for unconstrained thrust allocation for planar motion using standard damped inverse

Planar Motion - Linear Programming

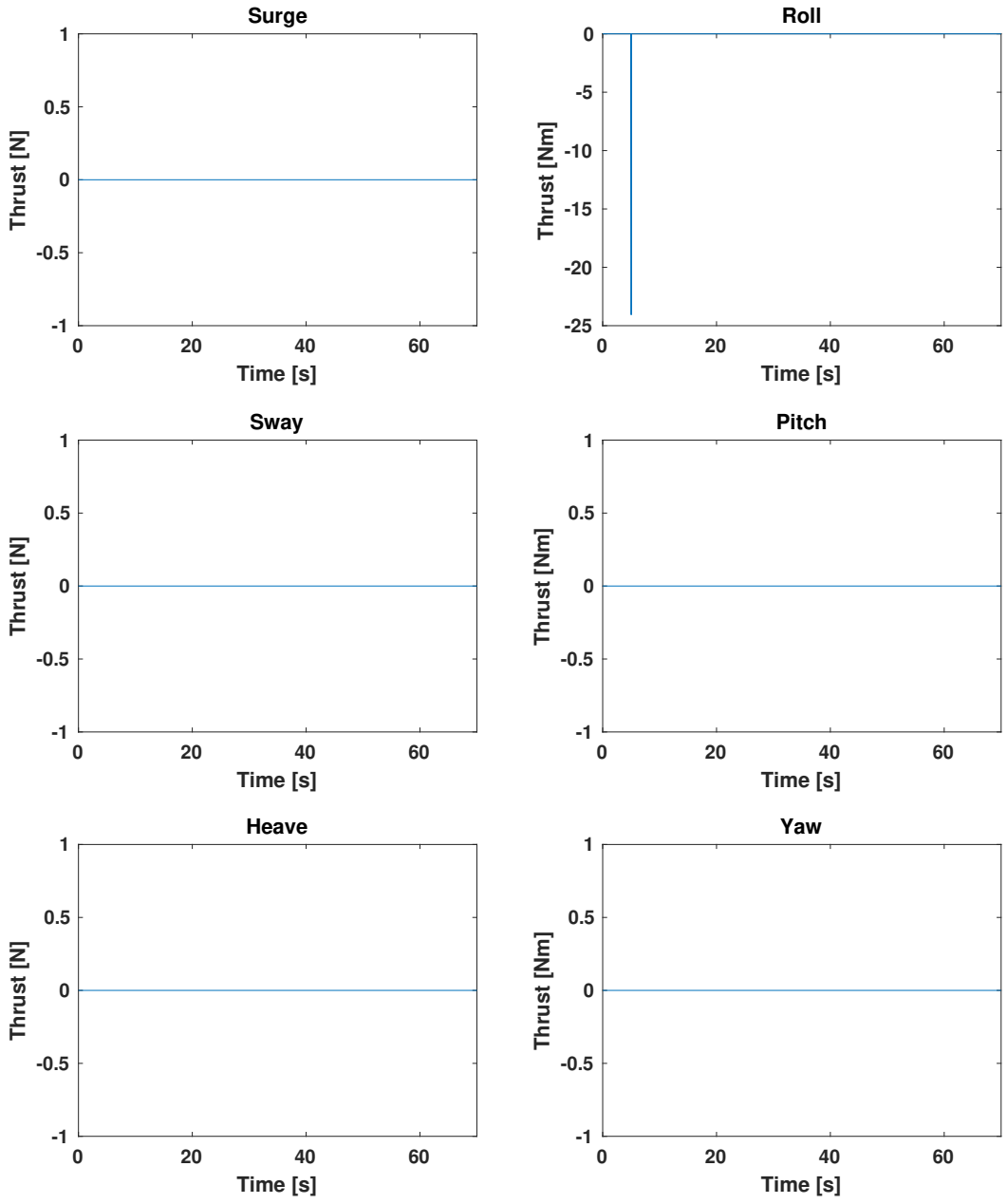


Figure 5.4: Error between commanded and actual thrust for unconstrained thrust allocation for planar motion using linear programming

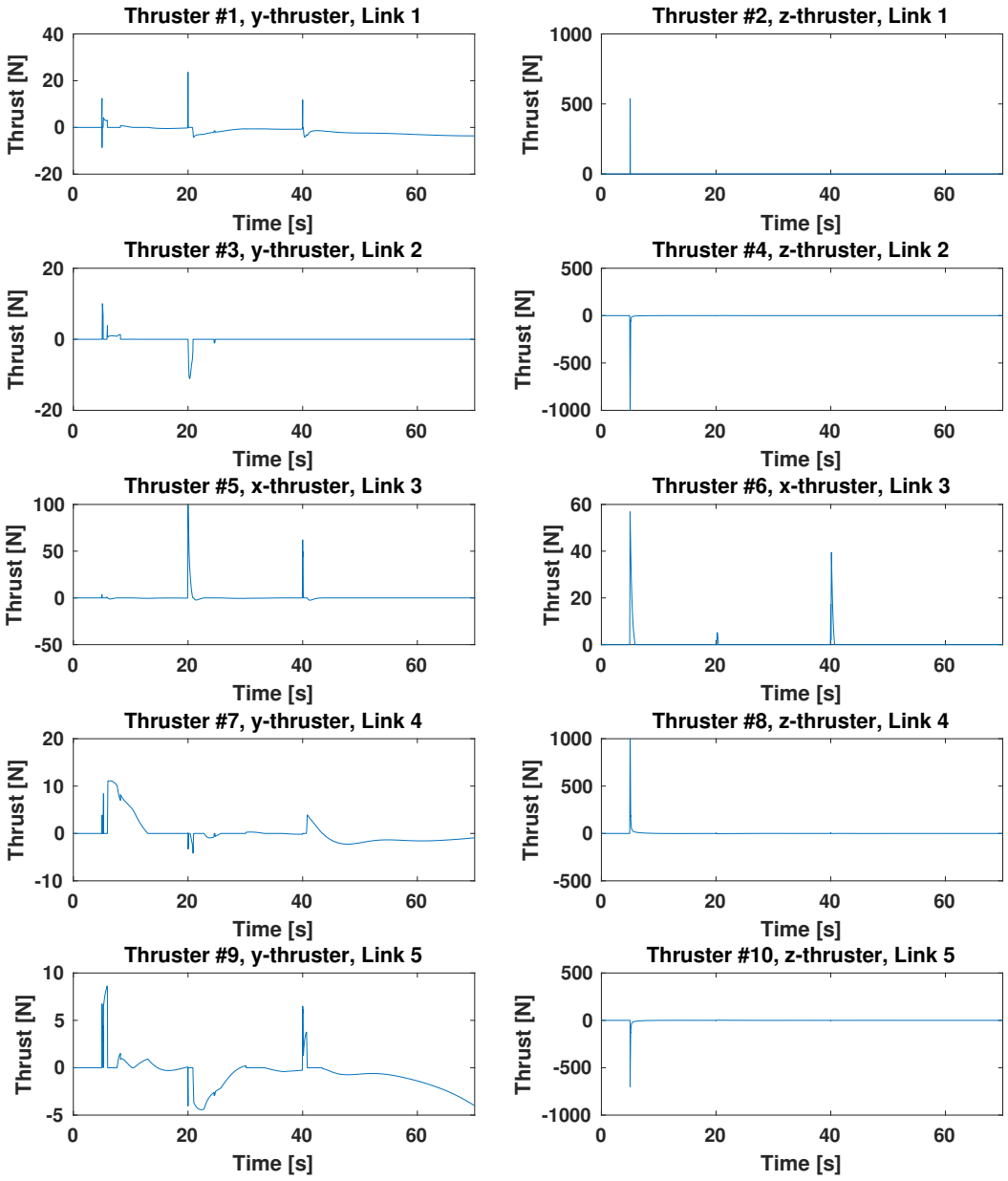


Figure 5.5: Individual thruster forces for unconstrained thrust allocation for planar motion using linear programming

Planar Motion - Quadratic Programming

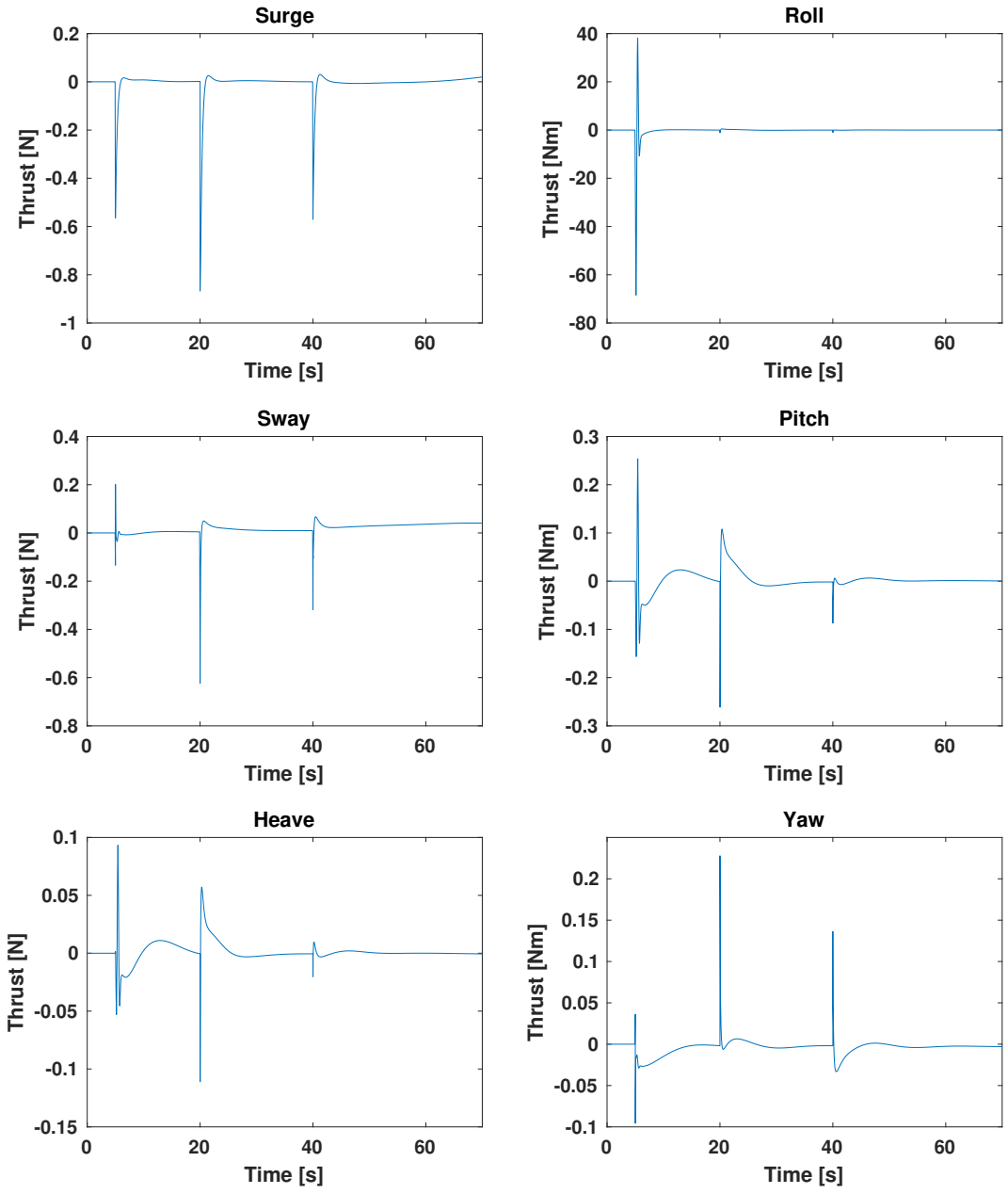


Figure 5.6: Error between commanded and actual thrust for unconstrained thrust allocation for planar motion using quadratic programming

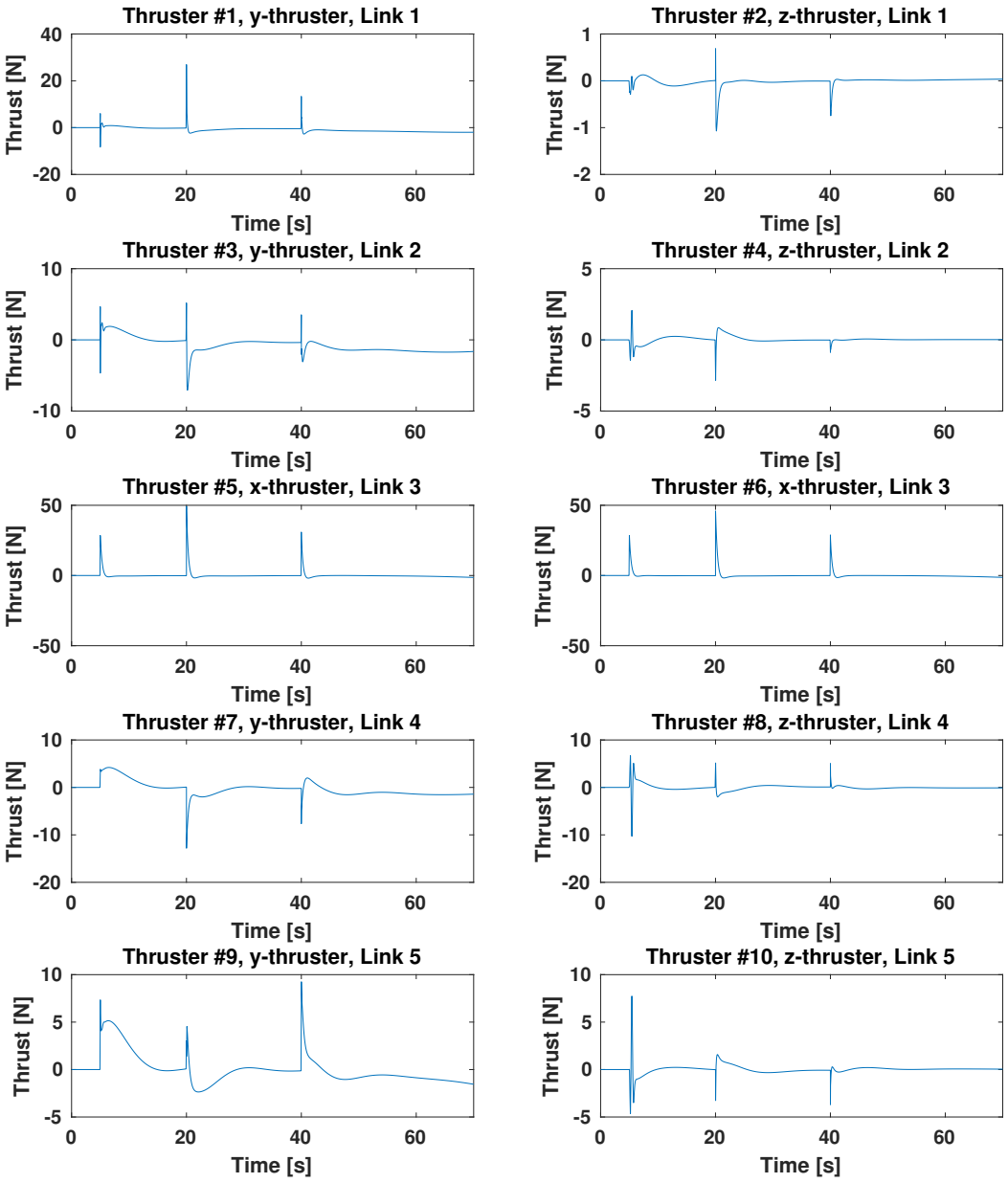


Figure 5.7: Individual thruster forces for unconstrained thrust allocation for planar motion using quadratic programming

Three-Dimensional Motion - Standard Damped Inverse

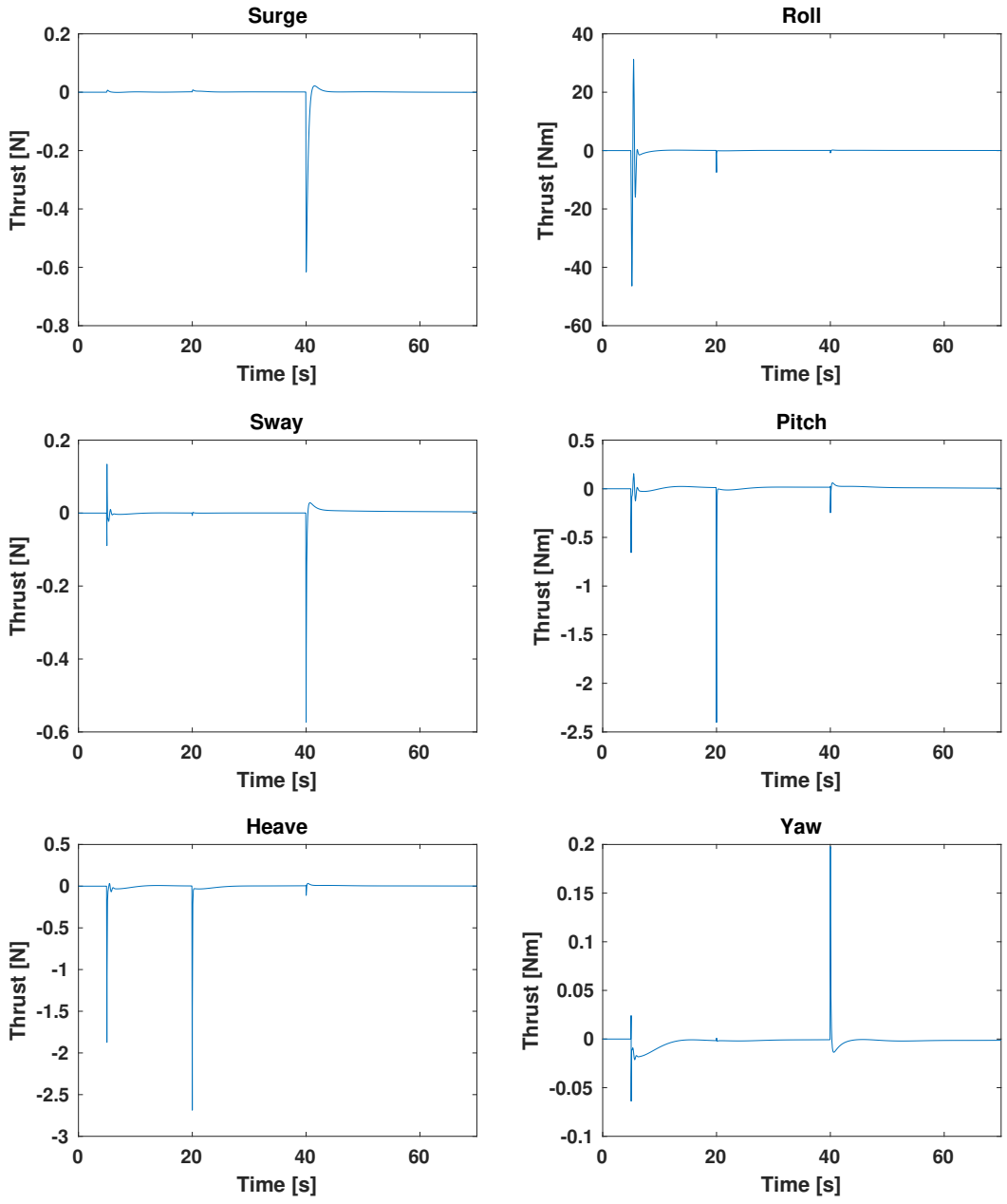


Figure 5.8: Error between commanded and actual thrust for unconstrained thrust allocation for three-dimensional motion using standard damped inverse

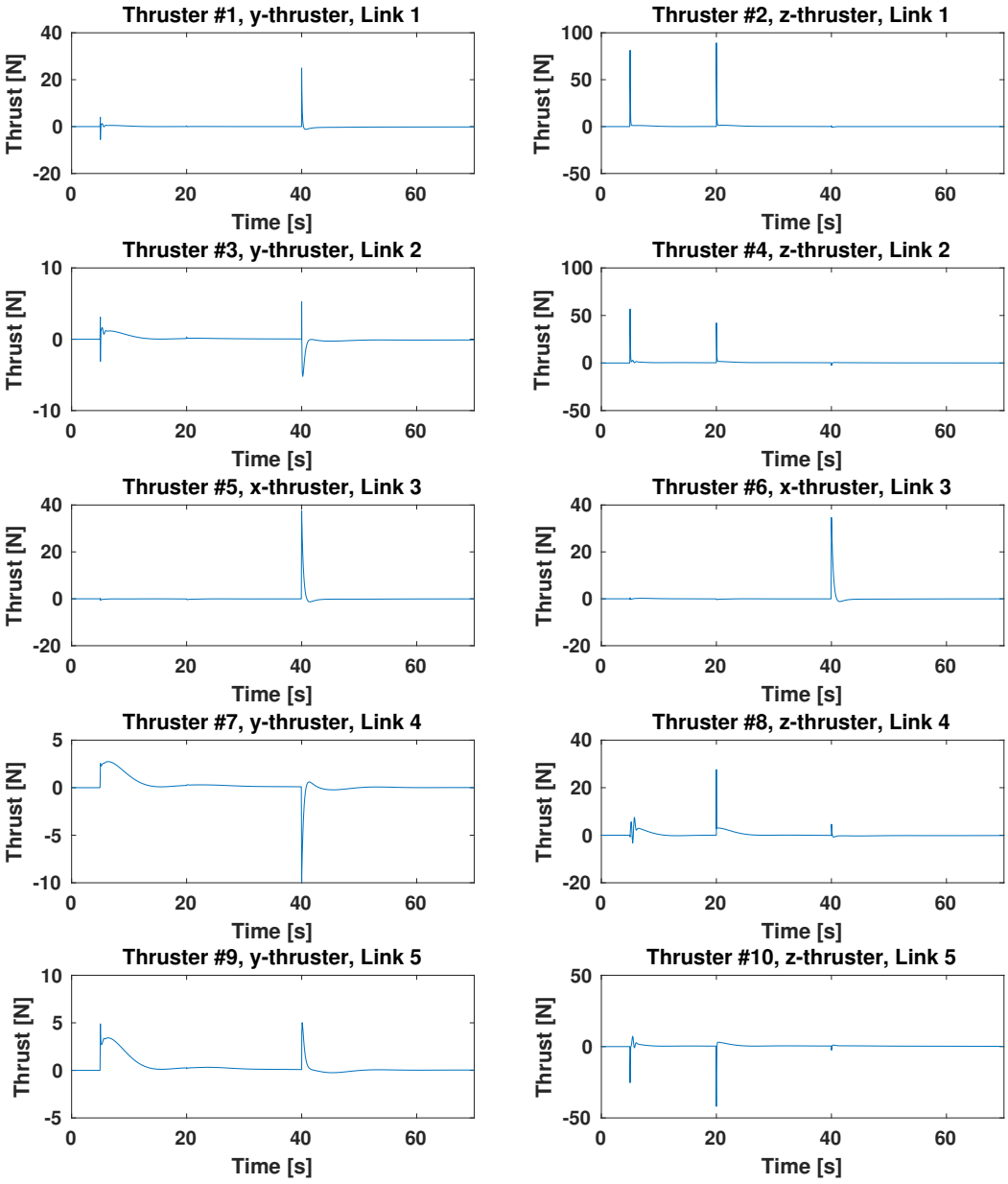


Figure 5.9: Individual thruster forces for unconstrained thrust allocation for three-dimensional motion using standard damped inverse

Three-Dimensional Motion - Linear Programming

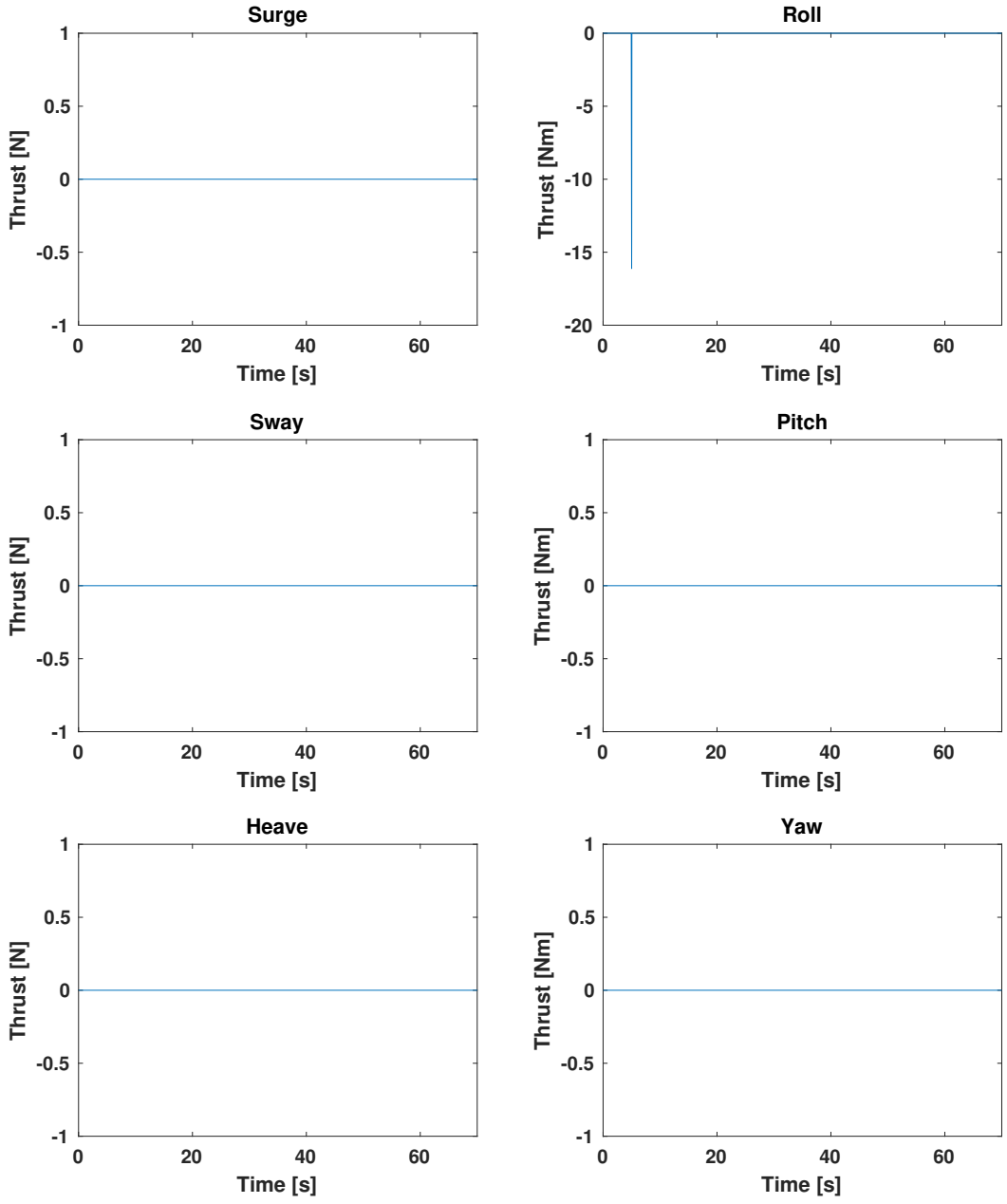


Figure 5.10: Error between commanded and actual thrust for unconstrained thrust allocation for three-dimensional motion using linear programming

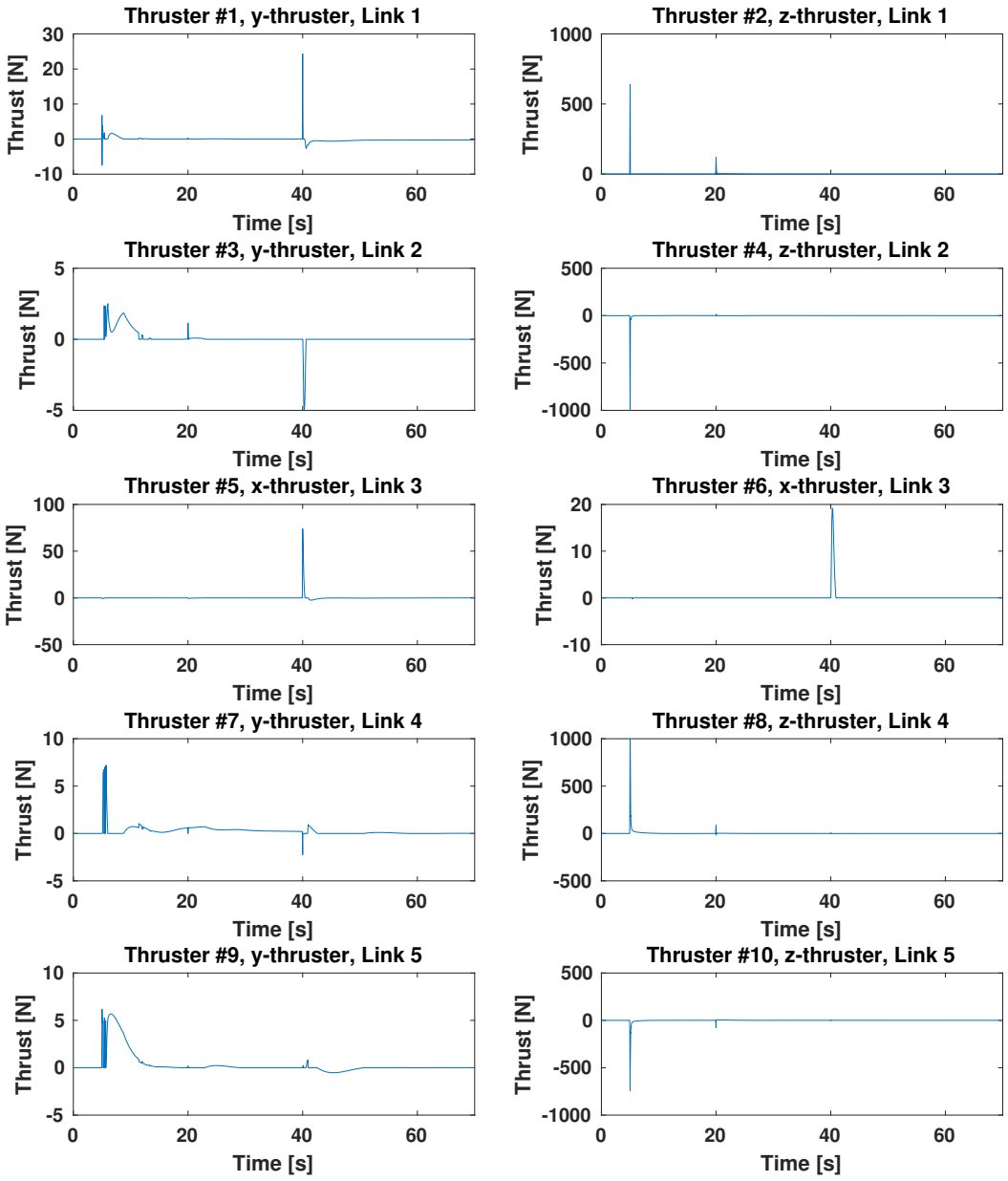


Figure 5.11: Individual thruster forces for unconstrained thrust allocation for three-dimensional motion using linear programming

Three-Dimensional Motion - Quadratic Programming

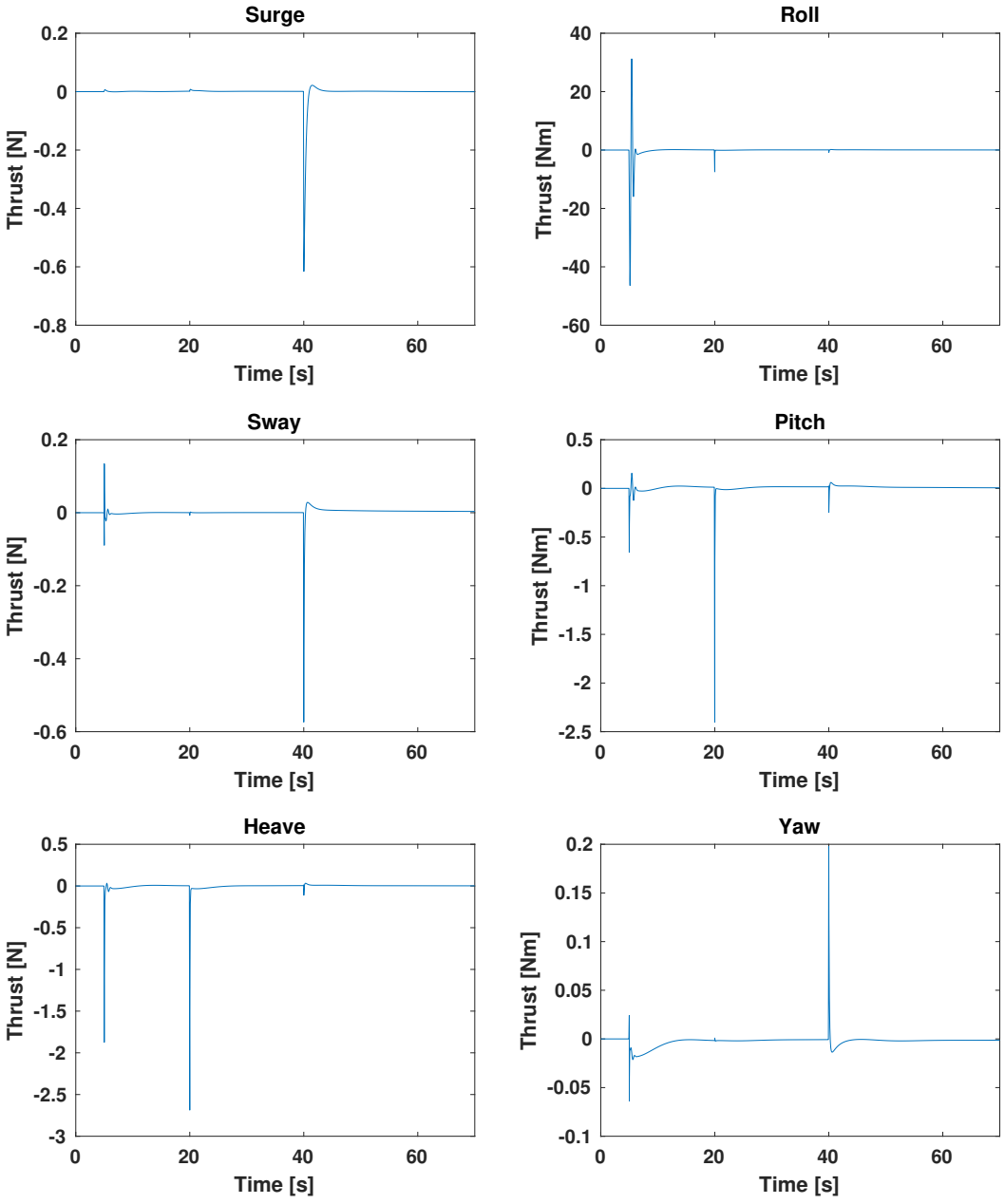


Figure 5.12: Error between commanded and actual thrust for unconstrained thrust allocation for three-dimensional motion using linear programming

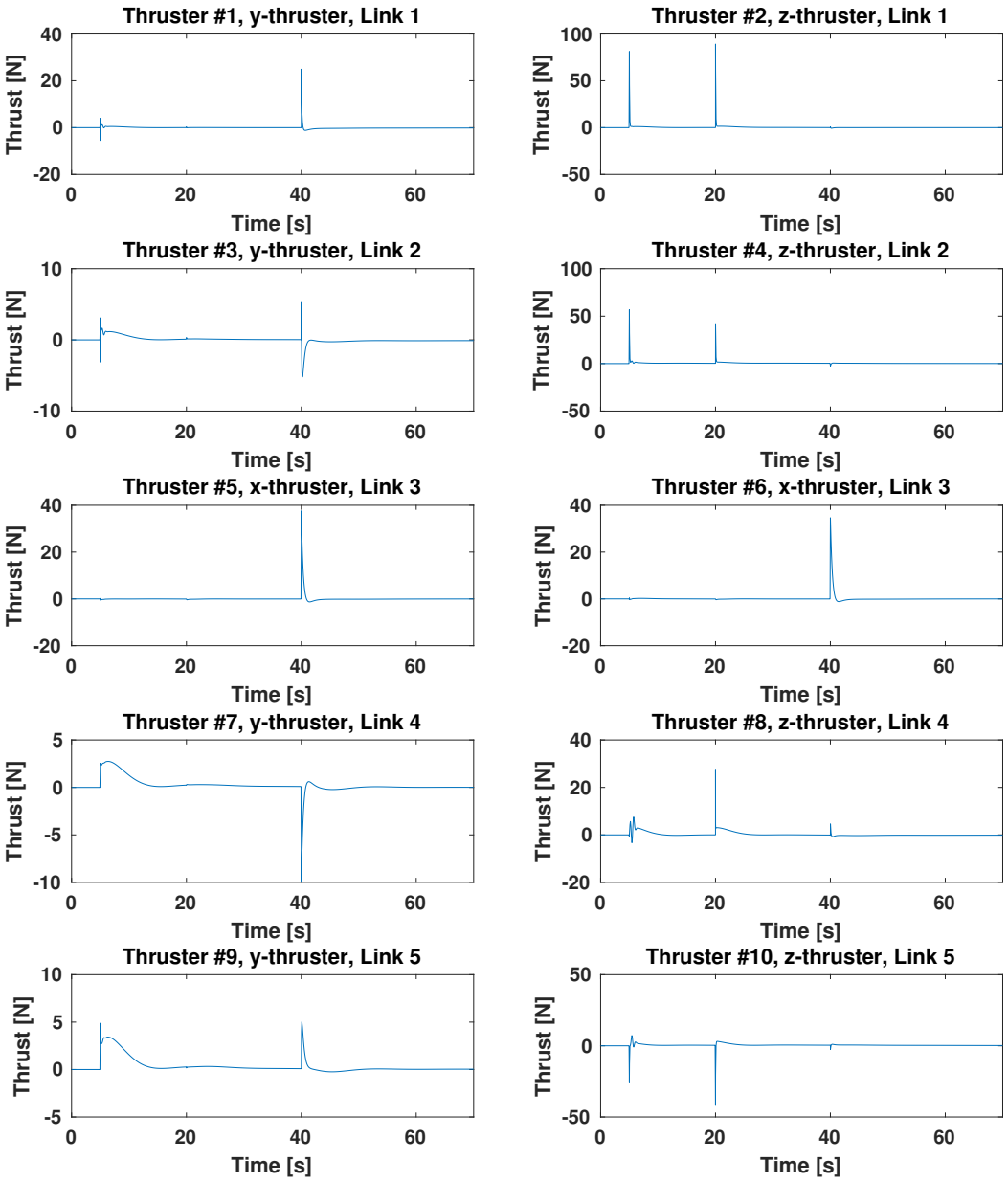


Figure 5.13: Individual thruster forces for unconstrained thrust allocation for three-dimensional motion using linear programming

5.2.2 Discussion

Firstly, from the maximum slack variable values (i.e. error) in Tab. 5.4, it can be concluded that all thrust allocation algorithms manage to allocate forces such that the commanded thrust is obtained with good precision. It can also be seen in Fig. 5.2, 5.4, 5.6, 5.8, 5.10 and 5.12 that the error is almost zero for most of the course of the simulation. From Fig. 5.1, it is evident that the end effector manages to follow the desired position. In Fig. 5.4 and Fig. 5.10, is can be observed that the error is equal to zero when using the LP algorithm for both planar and three-dimensional motion. The exception in both cases is the roll motion where the error has a spike. This is as expected due to the fact that the roll degree of freedom is not actuated, i.e. there are no thrusters producing thrust in roll. However, as a result of the design of the snake robot, the roll motion also converges to the initial state with time.

Although the LP algorithm manages to minimize the error to zero, it can be seen in Fig. 5.5 and Fig. 5.11, as well as in Tab. 5.5 that the force is poorly distributed amongst the thrusters. In fact, although a virtual thruster saturation limit of 1000 N and -1000 N is set for all thrusters, two of the thrusters actutually reach these limits. However, these thruster forces cancel each other out, which is the reason for the zero error. It should be noted that in real life, the thrusters would not achieve such a high force value as saturation and thruster dynamics would affect the thruster performance. Further tuning of the weight values in the LP algorithm might improve the performance and lower the maximum thrust values, although the cost would be a higher error. However, the high individual thruster forces are a known problem when using the LP algorithm. The optimal solutions of linear programs are found at the vertices of the feasible set. A consequence of this is that the algorithm tends to favor the use of smaller numbers of thrusters (Bodson (2002)). This is not ideal as wear and tear of the individual thrusters increase.

As opposed to the LP algorithm, the QP algorithm tends to use all thrusters, but to a smaller degree. This coincides with the results shown in Fig. 5.7 and Fig. 5.13, as well as the data in Tab. 5.5. As the QP algorithm manages to use all thrusters, while still maintaining low error values, its performance can be considered satisfactory. It can also be noted that the QP algorithm and the SDI algorithm produce the exact same thrust allocation results. This is expected for the unconstrained case when the damping coefficient ε in the SDI algorithm is equal to the inverse of the weight w in the QP algorithm. Under these conditions, the two algorithms can be mathematically deduced to each other (Fossen (2011)).

5.3 CASE 2: Constrained Thrust Allocation

In CASE 2, constrained thrust allocation is considered. The constraints implemented are thruster saturation limits, which are set to $u_{max,i} = 40$ N and $u_{min,i} = -40$ N for all individual thrusters. This makes the simulations more equal to real-life conditions as all physical thrusters have sat-

uration limits. Simulations are performed for both planar and three-dimensional motion using redistributed pseudo-inverse, linear programming and quadratic programming algorithms for thrust allocation.

5.3.1 Results

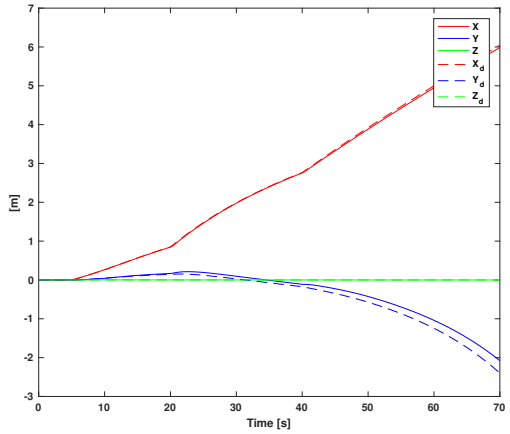
The simulation results for planar motion can be found in Fig. 5.15-5.18,as well as Fig. 5.14a, 5.14c and 5.14e. The simulation results for three-dimensional motion can be found in Fig. 5.21-5.24, as well as Fig. 5.14b, 5.14d and 5.14f.. Some simulation results are summed up in Tab. 5.6 and Tab. 5.7. Additional simulation results can be found App. A.2.

Table 5.6: The absolute value of the largest errors from simulations in CASE 2

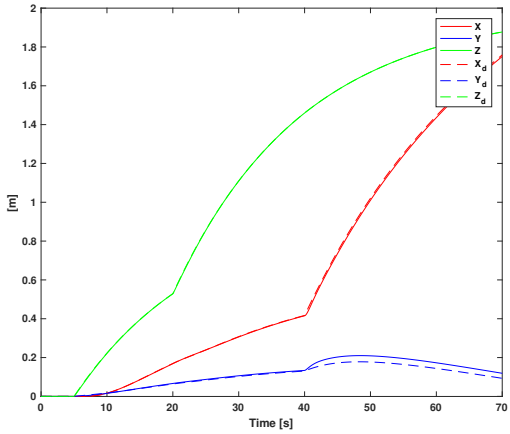
Motion	**Alloc. method**	**Surge [N]**	**Sway [N]**	**Heave [N]**	**Roll [Nm]**	**Pitch [Nm]**	**Yaw [Nm]**
2D	RPI	26.2463	53.4438	0.4452	68.4428	1.0350	144.9228
	LP	2.0385	0	0	59.5665	0	0
	QP	3.0993	1.3539	0.1124	68.4428	0.2641	0.4411
3D	RPI	0.6158	0.5735	114.6970	55.4292	199.5821	0.1978
	LP	0	0	44.9038	40.4113	0	0
	QP	0.6158	0.5736	39.7435	46.3857	14.3447	0.1978

Table 5.7: Maximum and minimum values for thruster forces from simulations in CASE 2

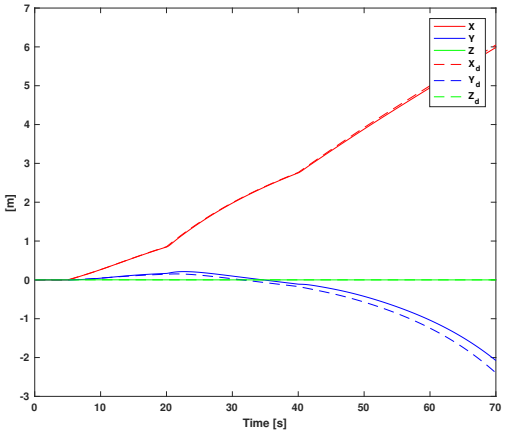
Motion	**Alloc. method**	**Value [N]**	**Thr. #1**	**Thr. #2**	**Thr. #3**	**Thr. #4**	**Thr. #5**	**Thr. #6**	**Thr. #7**	**Thr. #8**	**Thr. #9**	**Thr. #10**
2D	RPI	$u_{a,max}$	40.0000	3.0537	40.0000	2.0722	40.0000	40.0000	4.2088	20.5563	40.0000	7.7377
		$u_{a,min}$	-8.2498	-1.9612	-40.0000	-11.0180	-1.8836	-1.7015	-40.0000	-10.2962	-2.6017	-13.1528
	LP	$u_{a,max}$	40.0000	21.5429	7.4686	40.0000	40.0000	40.0000	11.1254	40.0000	29.5301	21.7850
		$u_{a,min}$	-10.0605	-21.8166	-11.0996	-40.0000	-2.4602	0	-40.0000	-40.0000	-4.4298	-27.9495
	QP	$u_{a,max}$	40.0000	0.7034	4.6576	2.0722	40.0000	40.0000	4.2088	6.6902	23.0544	7.7377
		$u_{a,min}$	-8.2498	-1.0753	-10.3459	-2.8621	-1.8843	-1.7414	-35.1841	-10.2962	-2.3617	-4.6535
3D	RPI	$u_{a,max}$	24.9679	40.0000	5.2618	40.0000	37.6404	34.6732	2.7391	40.0000	5.0226	7.2133
		$u_{a,min}$	-5.5030	-0.6479	-5.2066	-2.6328	-1.2880	-1.1650	-9.9931	-40.0000	-0.2437	-40.0000
	LP	$u_{a,max}$	28.5687	40.0000	2.6072	40.0000	40.0000	32.6012	7.1983	40.000	6.1921	26.9877
		$u_{a,min}$	-7.4567	-18.2657	-4.9883	-40.0000	-2.3755	-0.2815	-6.1545	-40.0000	-0.5286	-40.0000
	QP	$u_{a,max}$	24.9689	40.0000	5.2620	40.0000	37.6405	34.6731	2.7389	34.9545	5.0224	7.2700
		$u_{a,min}$	-5.5030	-0.6463	-5.2069	-2.6264	-1.2879	-1.1649	-9.9936	-15.1324	-0.2436	-40.0000



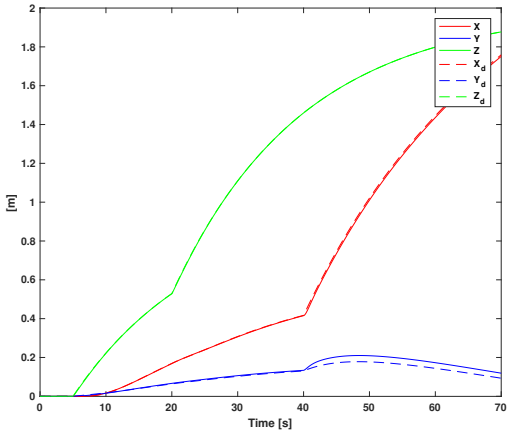
(a) 2D motion using SDI



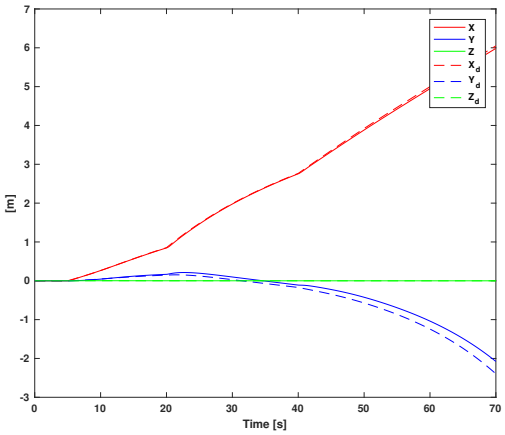
(b) 3D motion using SDI



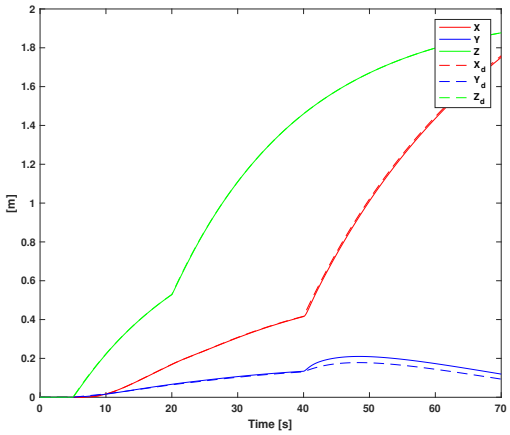
(c) 2D motion using LP



(d) 3D motion using LP



(e) 2D motion using QP



(f) 3D motion using QP

Figure 5.14: Position and desired position (dashed line) for end effector during CASE 2 simulations

Planar Motion - Standard Damped Inverse

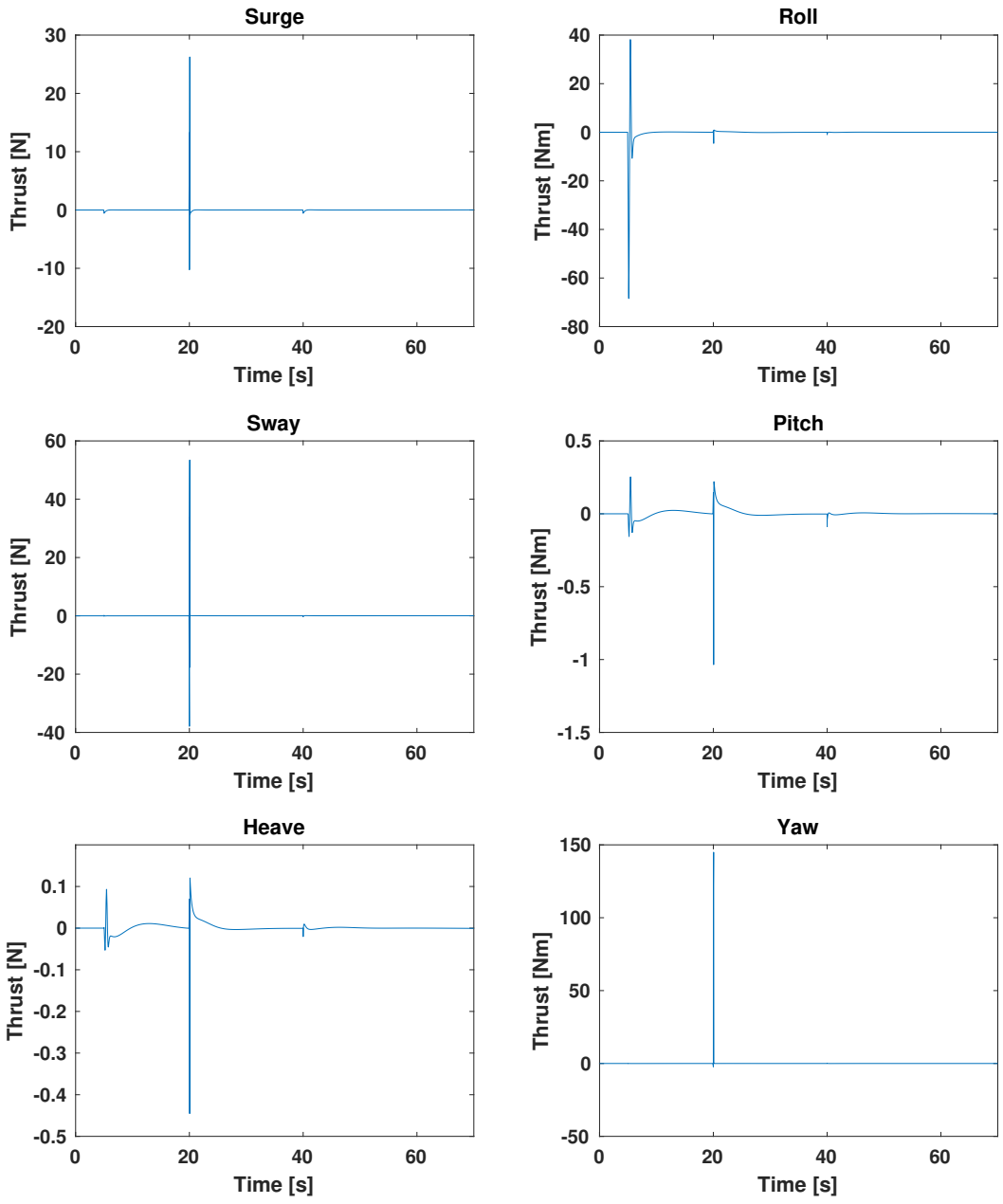


Figure 5.15: Error between commanded and actual thrust for constrained thrust allocation for planar motion using redistributed pseudo-inverse

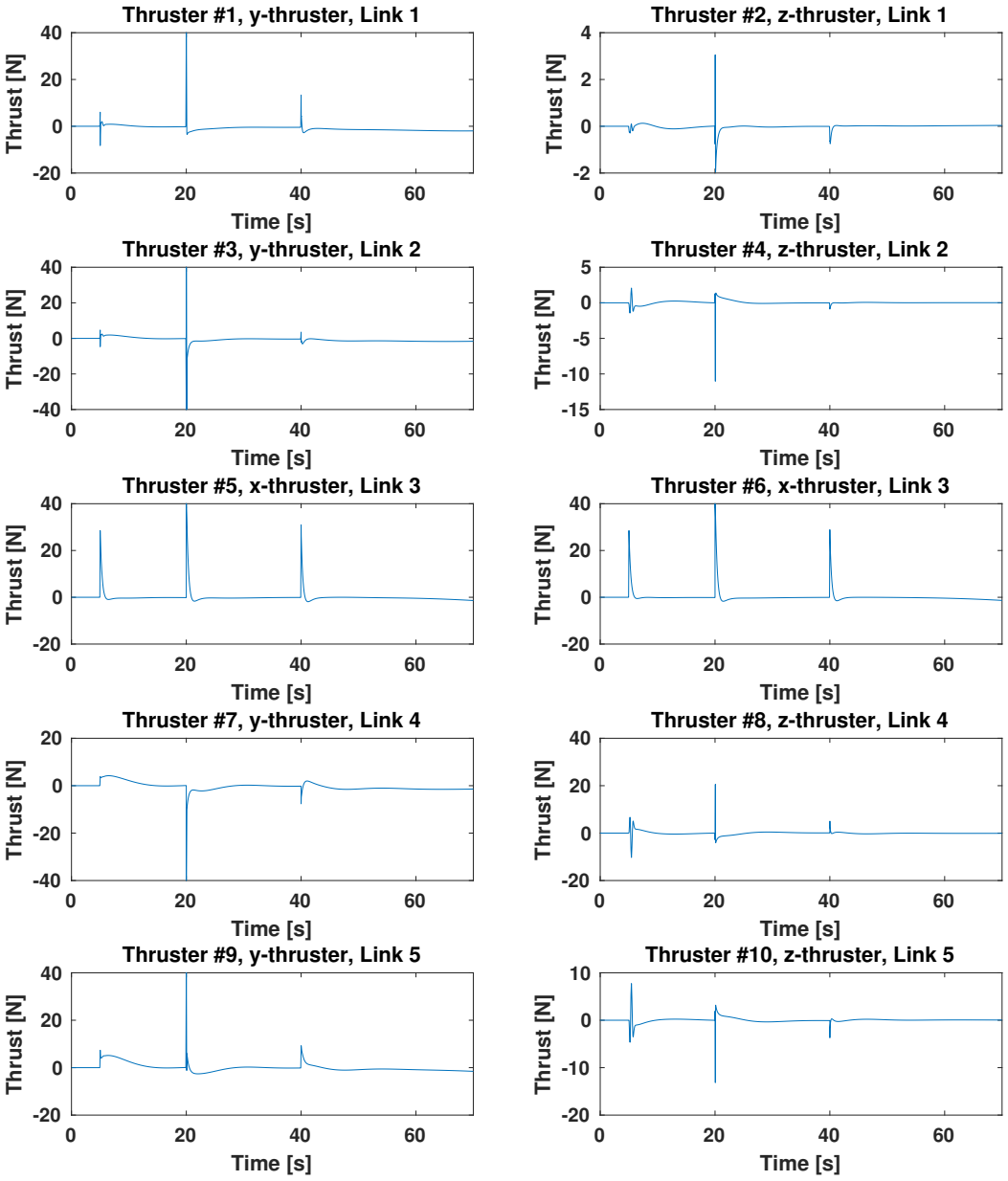


Figure 5.16: Individual thruster forces for constrained thrust allocation for planar motion using redistributed pseudo-inverse

Planar Motion - Linear Programming

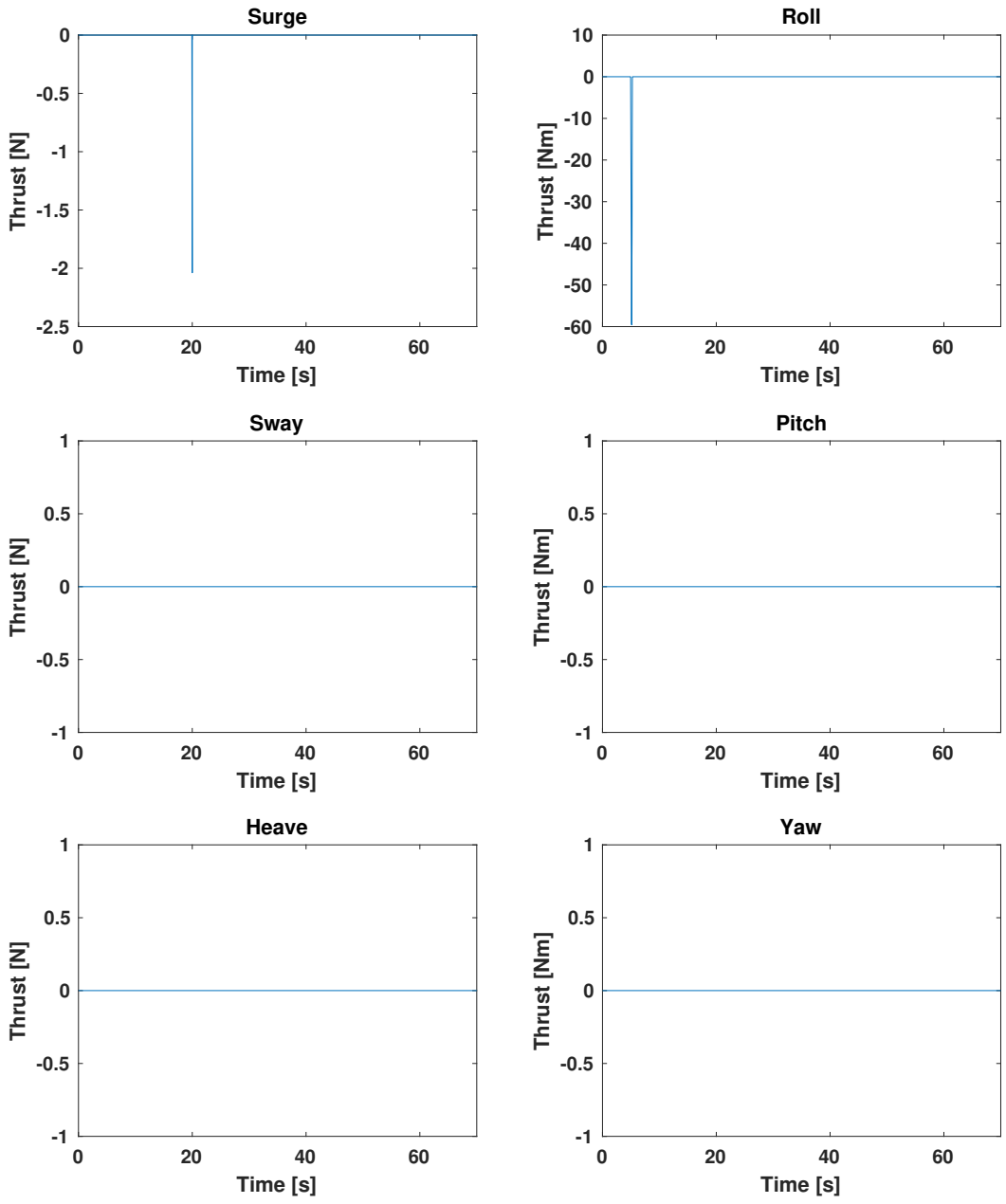


Figure 5.17: Error between commanded and actual thrust for constrained thrust allocation for planar motion using linear programming

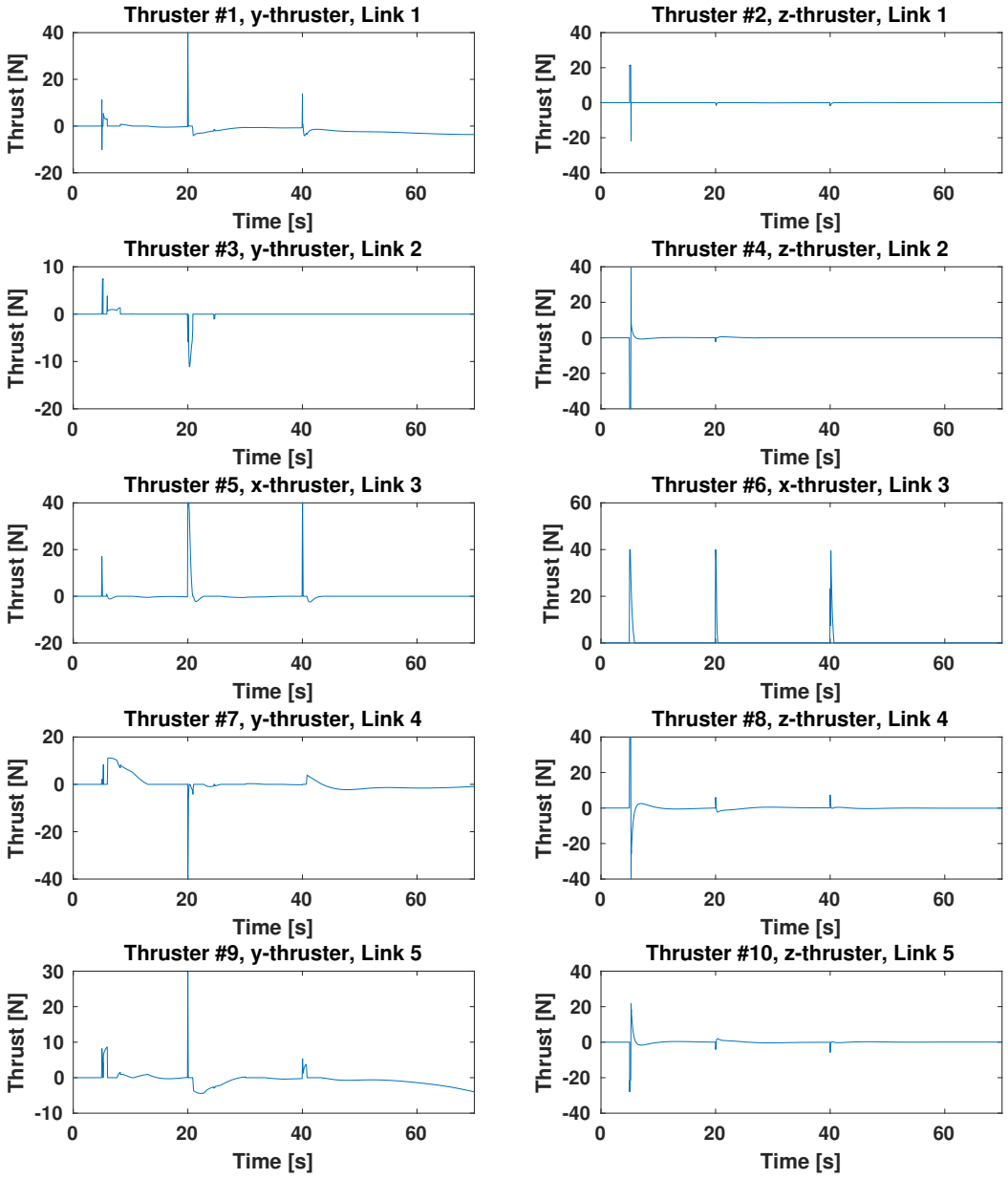


Figure 5.18: Individual thruster forces for constrained thrust allocation for planar motion using linear programming

Planar Motion - Quadratic Programming

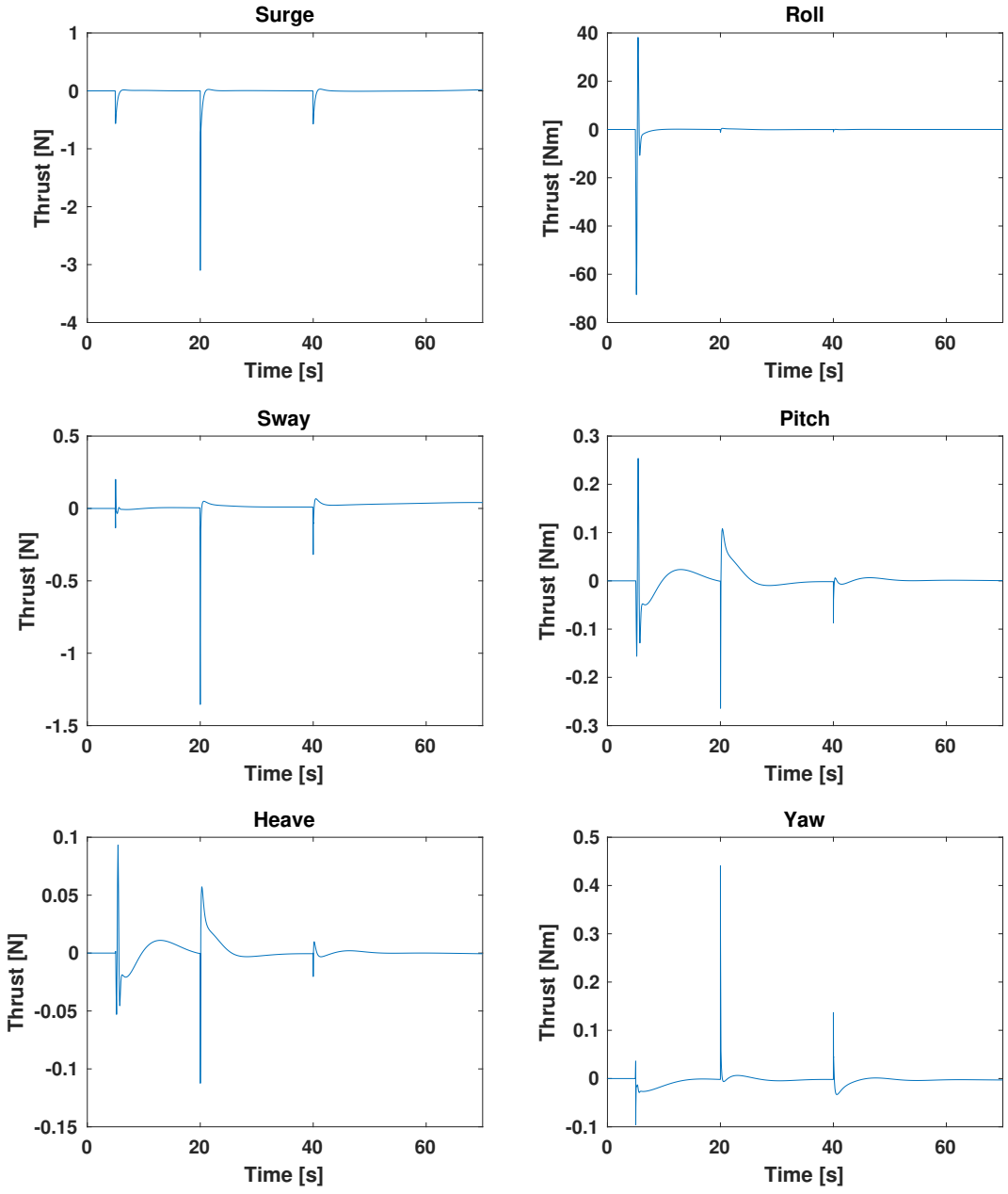


Figure 5.19: Error between commanded and actual thrust for constrained thrust allocation for planar motion using quadratic programming

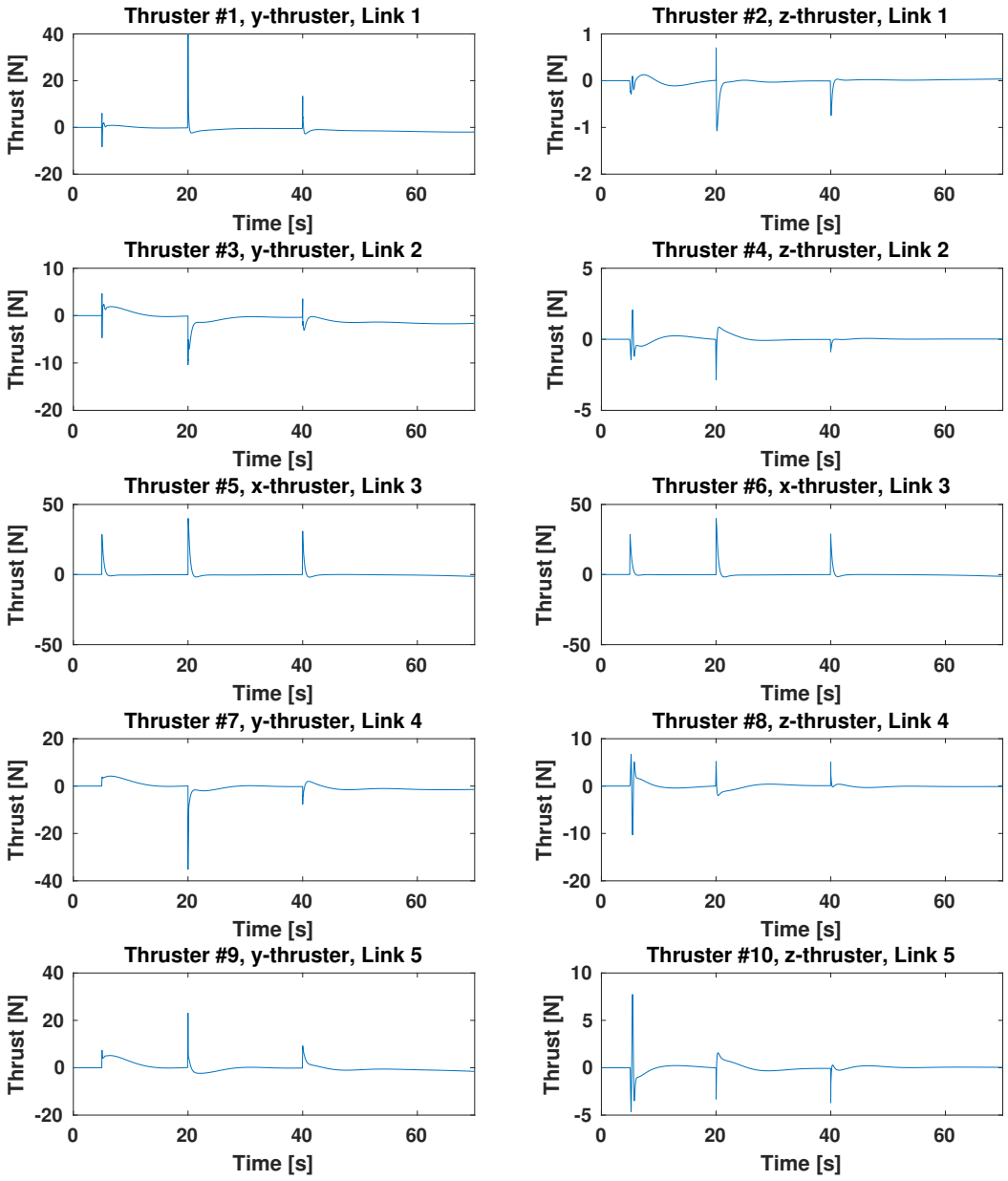


Figure 5.20: Individual thruster forces for unconstrained thrust allocation for planar motion using quadratic programming

Three-Dimensional Motion - Redistributed Pseudo-Inverse

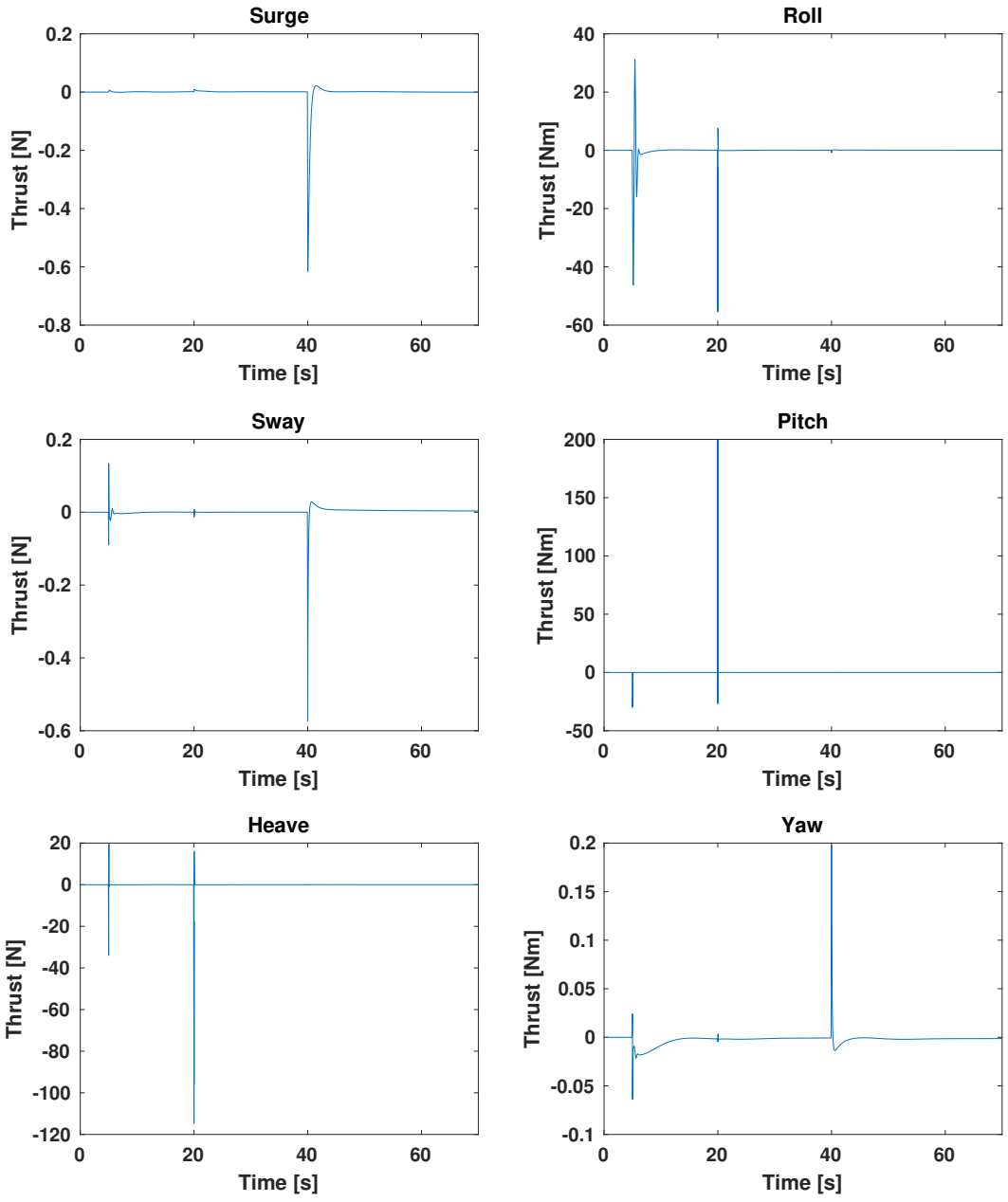


Figure 5.21: Error between commanded and actual thrust for constrained thrust allocation for three-dimensional motion using redistributed pseudo-inverse

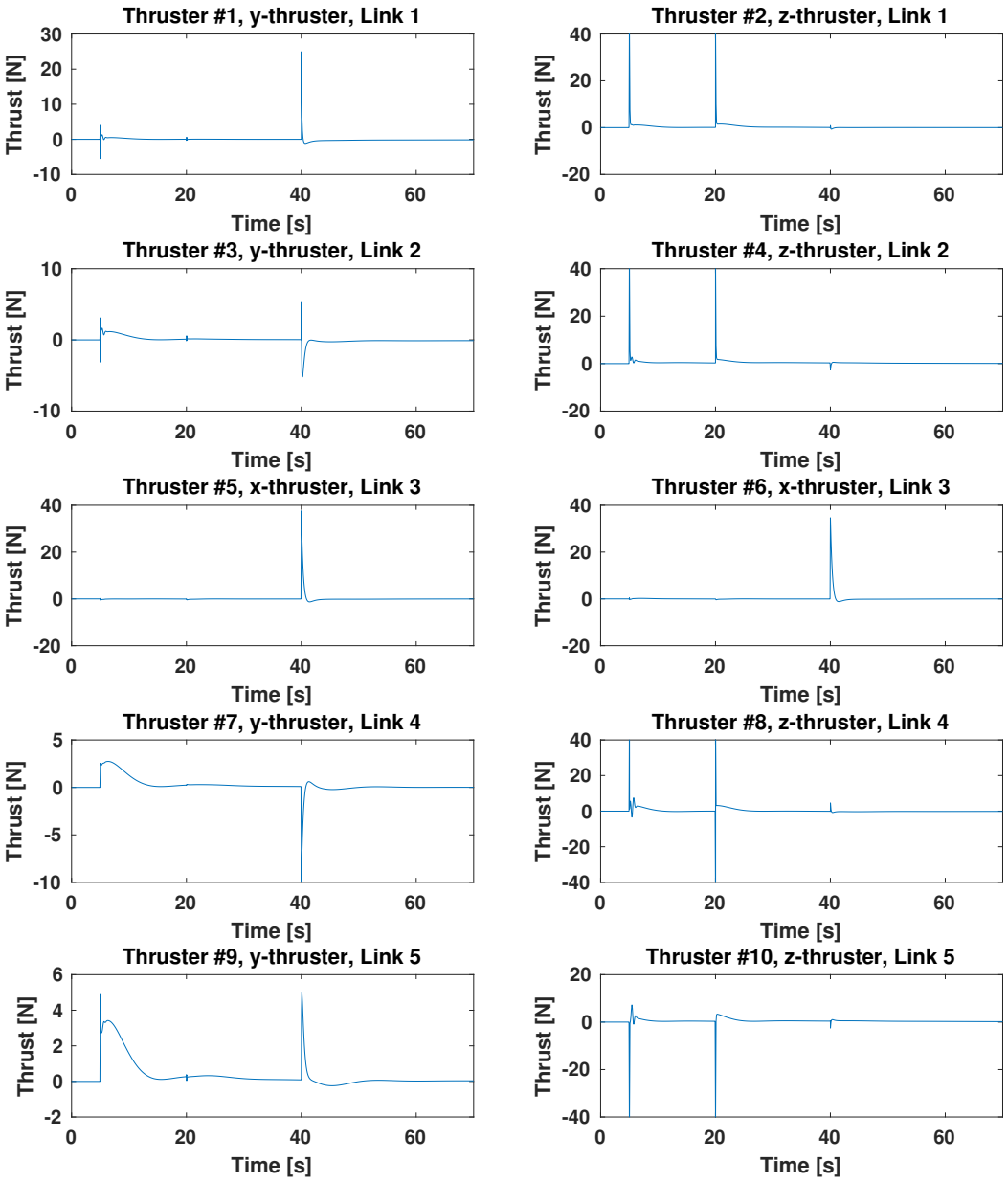


Figure 5.22: Individual thruster forces for constrained thrust allocation for three-dimensional motion using redistributed pseudo-inverse

Three-Dimensional Motion - Linear Programming

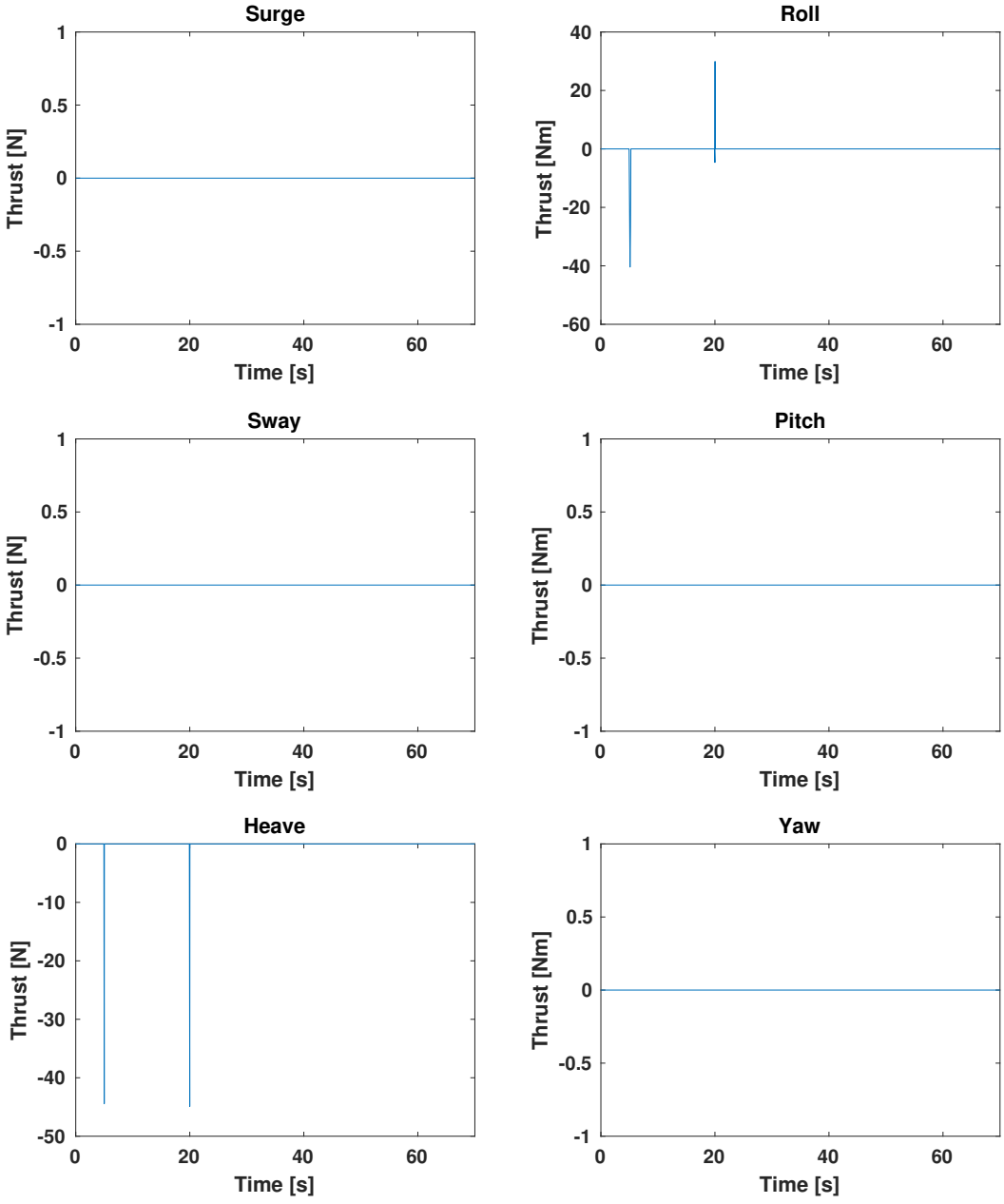


Figure 5.23: Error between commanded and actual thrust for constrained thrust allocation for three-dimensional motion using linear programming

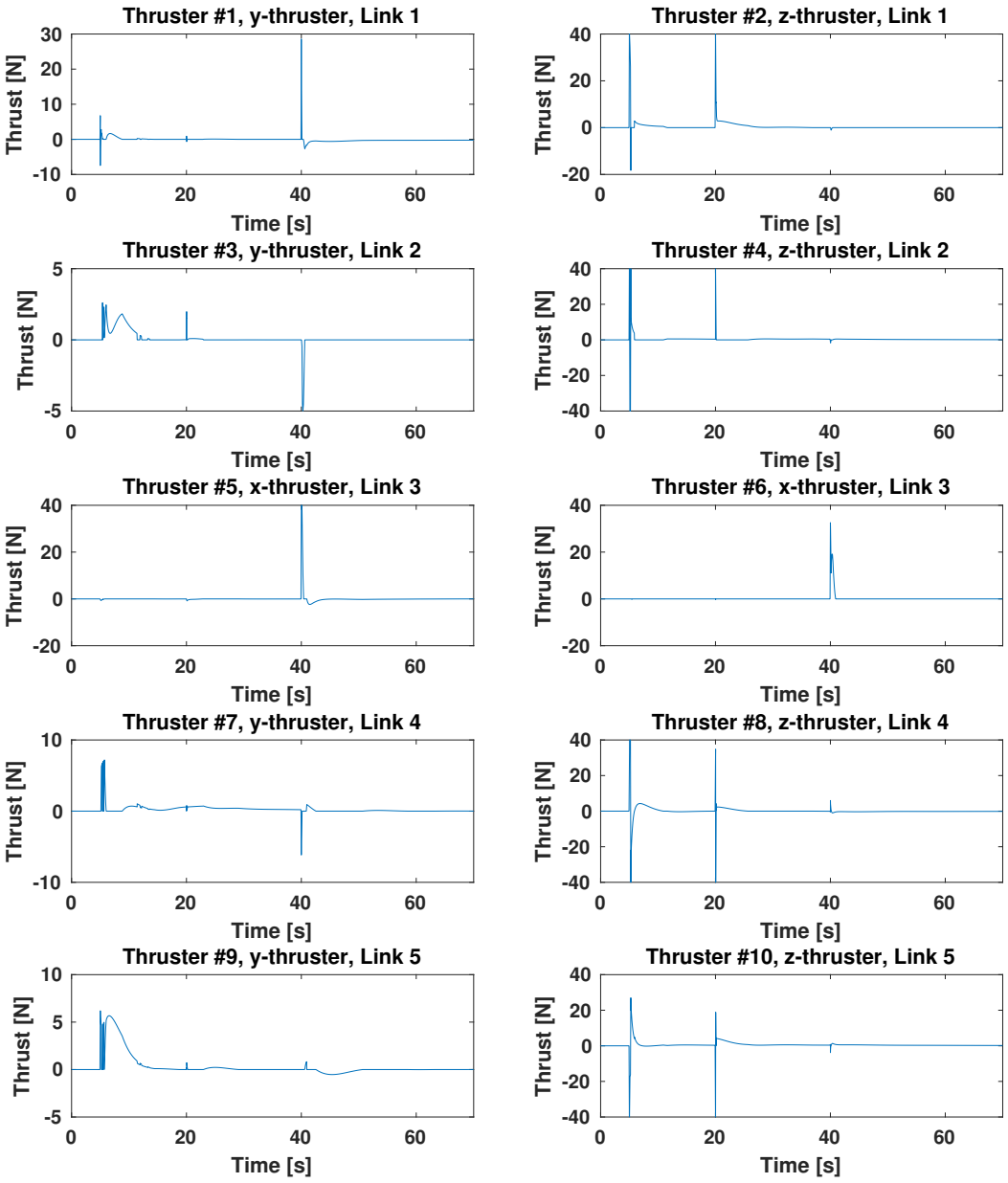


Figure 5.24: Individual thruster forces for constrained thrust allocation for three-dimensional motion using linear programming

Three-Dimensional Motion - Quadratic Programming

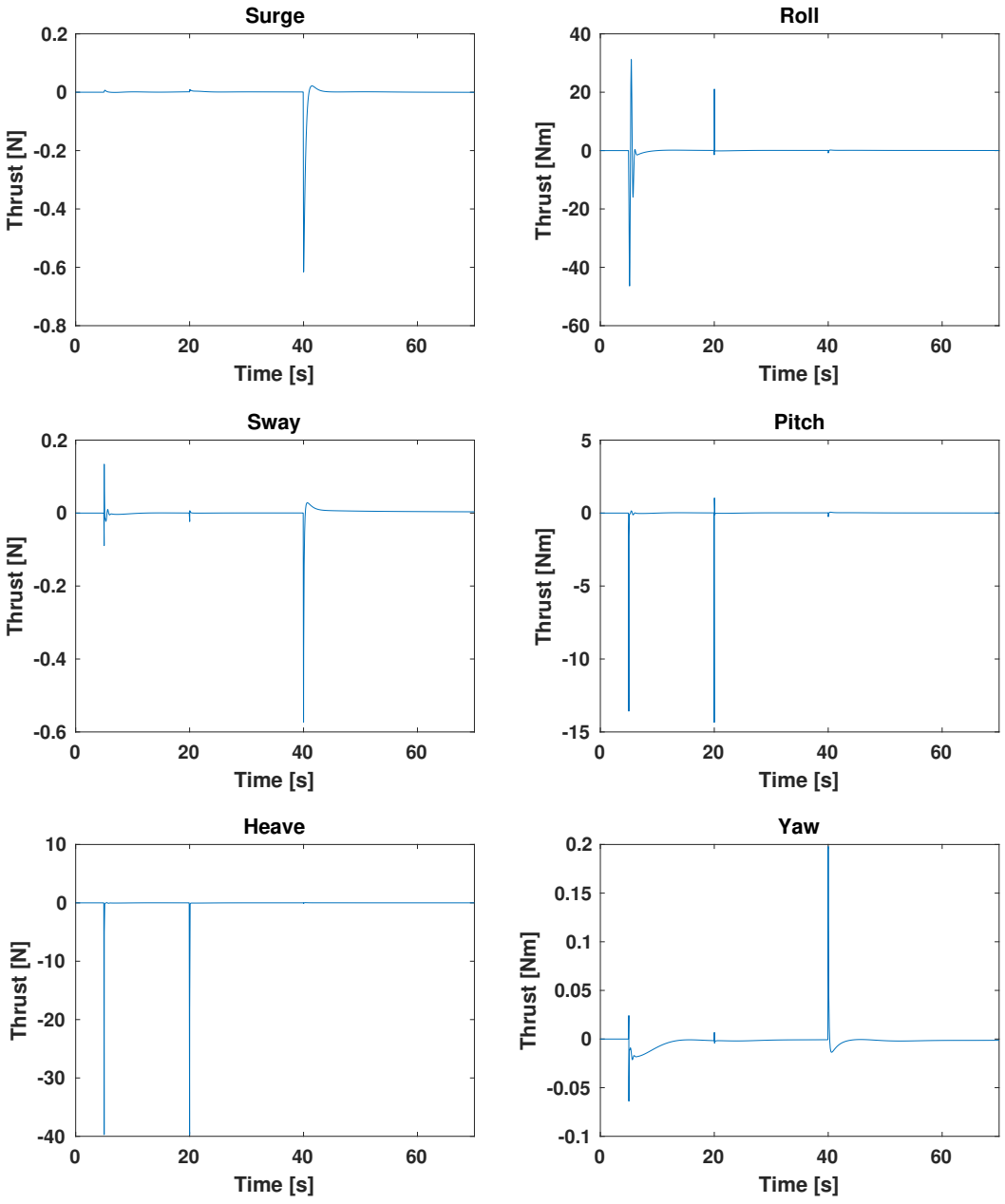


Figure 5.25: Error between commanded and actual thrust for constrained thrust allocation for three-dimensional motion using quadratic programming

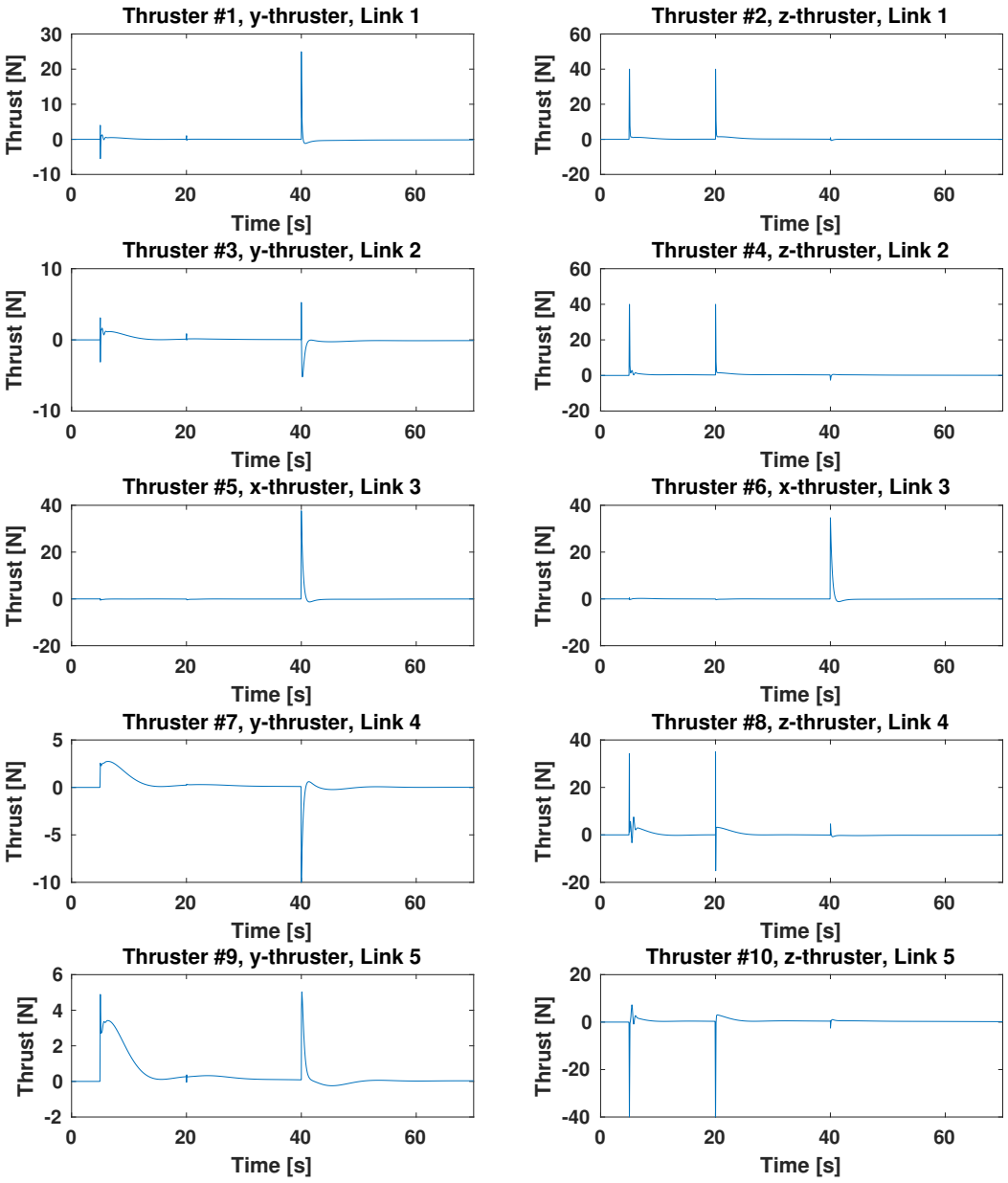


Figure 5.26: Individual thruster forces for constrained thrust allocation for three-dimensional motion using quadratic programming

5.3.2 Discussion

Also in CASE 2, it can be seen from Fig. 5.15, 5.17, 5.19, 5.21, 5.23 and 5.25, as well as the position plots in Fig. 5.14, that all thrust allocation algorithms manage to fulfill their purpose of allocating thrust forces such that the commanded thrust is obtained. However, the maximum error values in Tab. 5.6 are significantly larger than the ones in Tab. 5.4. This is as expected for the constraint allocation case as commanded thrust that can not be allocated due to saturated thrusters is accounted for by the slack variable.

When using the RPI algorithm, it can be seen in Tab. 5.6 that the extreme error values for planar motion is large compared to the extreme values for three-dimensional motion. Also, from Tab. 5.7 and Fig. 5.16 and 5.22, it can be observed that the unsaturated thrusters produce low thrust in the planar motion simulation, compared to the simulation of three-dimensional motion. This suggests that the performance of the algorithm may vary greatly dependent on the task to be performed.

For the simulations using the LP algorithm, it can be seen in Fig. 5.17 and 5.23, that the error is equal to zero for the most of the course of the simulation, as in CASE 1. In Fig. 5.17, it can be noticed that the error has a spike in surge. From Fig. 5.18, it can be observed that both thruster number five and thruster number six, which can be assumed to have the largest contribution to the global surge thrust, are saturated. This suggests that the spike in error is due to a lack of enough x-direction thrusters. There are also spikes in error in the results from the three-dimensional motion simulation, as shown in Fig. 5.23, which can be assumed to be due to the same reason. However, Fig. 5.18 and 5.24, as well as the results in Tab. 5.7, show that the LP algorithm distributes force to thrusters more evenly in the constrained case than it did in the unconstrained case.

When comparing the results in Tab. 5.7 for thrust allocation using QP with the results for thrust allocation using RPI and LP, it can be noticed that fewer thrusters reach saturation, although the results in Tab. 5.6 show that the error is small for most generalized DOFs. This suggests that the QP algorithm manages the task of distributing forces well. However, for the tree-dimensional case, some errors in heave and pitch are still large. This might be because the implemented USR only has z-revolute joints. Therefore, all vertical motion must be produced by thrusters, which causes the thruster force demand relative to displacement to be higher than it is for planar motion.

Although it is out of the scope of this thesis, it should also be mentioned that a substantial challenge concerning USRs lies in the inverse kinematics and controller models. A dynamic inverse kinematics model and a dynamic controller model might be stable separately. However, getting the two to work well together can prove a challenging endeavor. A smoother controller output would be preferable when analyzing the performance of thrust allocation algorithms.

Chapter 6

Concluding Remarks

6.1 Conclusion

The topic of this master thesis was to study thrust allocation algorithms for USRs using methods such as redistributed pseudo-inverse, linear programming and quadratic programming. A study of previous literature concerning snake robots, underwater snake robots and control allocation was performed. Also, a thorough description of the underwater snake robot simulation model used in this thesis, was presented. Several thrust allocation methods, both explicit solution methods and iterative methods, were presented and described.

The algorithms were developed and implemented in Matlab/SIMULINK. Simulations were performed using the underwater snake robot simulation model for several cases simulating unconstrained and constrained thrust allocation for planar and three-dimensional motion. In the unconstrained case, simulations using a pre-implemented standard damped inverse algorithm was also performed for comparison. A redistributed pseudo-inverse algorithm was developed and implemented for simulation of constrained thrust allocation.

From the simulation results, it could be concluded that all methods are viable for thrust allocation of USRs. All methods managed to solve the thrust allocation problem in such a way that the error between commanded and actual thrust was generally kept to a minimum. For unconstrained thrust allocation, the performance of the QP algorithm proved to be equal to the performance of the pre-implemented SDI algorithm. The LP algorithm tended to favor allocation to a small number of thrusters, which is a drawback considering thruster wear and tear. However, the result also showed that the LP algorithm generally kept the error exceptionally low in all actuated DOFs.

The redistributed pseudo-inverse algorithm implemented for constrained allocation proved to have a sub-optimal performance with high error values compared to the other methods. In comparison, both the LP algorithm and the QP algorithm gave better results. The LP algorithm had a better performance for constrained thrust allocation than it did in the unconstrained case,

since it was forced to allocate between a larger number of thrusters when some thrusters got saturated. The best performance was obtained when using the QP algorithm, which had the most even allocation between individual thrusters while still keeping the error low. For QP, the performance was also high when simulating three-dimensional motion, compared to the other algorithms.

6.2 Further Work

In order to obtain more realistic simulation results, thruster dynamics should be implemented in the simulation model. Also, additional constraints such as rate limits and forbidden sectors should be added to the algorithms. A rate limit would prevent the thruster forces from unrealistically spiking, and forbidden sectors would prevent thruster forces from counteracting other out.

Simulations should be performed using several snake robot configurations, i.e. different combinations of link and joint types. Simulations of more complex motions, such as intervention maneuvers and transit mode motion, should also be performed in order to obtain a more thorough basis for algorithm evaluation.

Bibliography

Bauchot, R. (1994). *Snakes: A Natural History*. Sterling Publishing Company, New York, USA.

Berge, S. P. and Fossen, T. I. (1997). Robust control allocation of overactuated ships: Experiments with a model ship. In *Proc. of the 4th IFAC Conf. on Manoeuvring and Control of Marine Craft*, pages 166–171, Brijuni, Croatia.

Bloch, A. M., Baillieul, J., Crouch, P., and Marsden, J. (2003). *Nonholonomic Mechanics and Control*. Springer-Verlag, New York.

Bodson, M. (2002). Evaluation of optimization methods for control allocation. *Journal of Guidance, Control and Dynamics*, 25(4):703–711.

Bodson, M. and Frost, S. A. (2011). Load balancing in control allocation. *Journ. of Guidance, Control, and Dynamics*, 34(2):380–387.

Bordignon, K. A. and Durham, W. C. (1995). Closed-form solutions to constrained control allocation problem. *Journ. of Guidance, Control, and Dynamics*, 18(5):1000–1007.

Boyer, F., Porez, M., and Khalil, W. (2006). Macro-continuous computed torque algorithm for a three-dimensional eel-like robot. *IEEE Trans. on Robotics*, 22(4):763–775.

Buffington, J. M. and Enns, D. F. (1996). Lyapunov stability analysis of daisy chain control allocation. *Journ. of Guidance, Control, and Dynamics*, 19(6):1226–1230.

Chen, L., Wang, Y., Ma, S., and Duan, D. (2007). *Study on Locomotion of a Crawling Robot for Adaption to the Environment*, chapter 18, pages 301–316. I-Tech Education and Publishing.

Chernousko, F. L. (2005). Modelling of snake-like locomotion. *Applied Mathematics and Computation*, 164(2):415–434.

Crespi, A., Badertscher, A., Guignard, A., and Ijspeert, A. J. (2005). AmphiBot I: an amphibious snake-like robot. *Robotics and Autonomous Systems*, 50(4):163–175.

Crespi, A. and Ijspeert, A. J. (2006). AmphiBot II: amphibious snake robot that crawls and swims using a central pattern generator. In *Proc. 9th Int. Conf. on Climbing and Walking Robots (CLAWAR)*, pages 19–27, Brussels, Belgium.

Date, H. and Takita, Y. (2005). Control of 3d snake-like locomotive mechanism based on continuum modeling. In *Proc. ASME 2005 International Design Engineering Technical Conferences*, DETC2005-85130.

Durham, W. C. (1993). Constrained control allocation. *Journal of Guidance, Control and Dynamics*, 16(4):717–725.

Endo, G., Togawa, K., and Hirose, S. (1999). Study on self-contained and terrain adaptive active cord mechanism. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, volume 3, pages 1399–1405.

Faltinsen, O. M. (1990). *Sea Loads on Ships and Offshore Structures*. Cambridge University Press.

Fossen, T. I. (1994). *Guidance and Control of Ocean Vehicles*. John Wiley & Sons, Inc., Chichester.

Fossen, T. I. (2011). *Handbook of Marine Craft Hydrodynamics and Motion Control*. John Wiley & Sons Ltd., West Sussex.

Fossen, T. I. and Johansen, T. A. (2006). A survey of control allocation methods for ships and underwater vehicles. In *14th Mediterranean Conf. on Control and Automation*, pages 1–6, Ancona, Italy.

Golub, G. H. and Loan, C. F. (1983). *Matrix computations*. North Oxford Academic, Oxford.

Grabec, I. (2002). Control of creeping snake-like robot. In *Proc. 7th Int. Workshop on Advanced Motion Control*, pages 526–513.

Grey, J. (1946). The mechanism of locomotion in snakes. *Journal of Experimental Biology*, 23(2).

Hicks, G. P. (2003). *Modeling and control of a snake-like serial link structure*. Ph.d. dissertation, North Carolina State University.

Hirose, S. (1993). *Biologically Inspired Robots: Snake-Like Locomotors and Manipulators*. Oxford University Press, Oxford.

Hirose, S. and Yamada, H. (2009). Snake-like robots. *IEEE Robotics & Automation Magazine*, 16(1):88–98.

Hu, D. L., Nirody, J., Scott, T., and Shelley, M. J. (2009). The mechanics of slithering locomotion. *Proc. National Academy of Sciences, USA*, 106:10081–10085.

Härkegård, O. (2002). Efficient active set algorithms for solving constrained least squares problems in aircraft control allocation. In *Proc. of the 41st IEEE Conf. on Decision and Control*, pages 1295–1300, Las Vegas, NV USA.

Johansen, T. A. and Fossen, T. I. (2013). Control allocation - a survey. *Automatica*, 49(5):1087–1103.

Kamegawa, T., Harada, T., and Gofuku, A. (2009). Realization of cylinder climbing locomotion with helical form by snake robot with passive wheels. In *IEEE Int. Conf. on Robotics and Automation*, pages 3067–3072.

Kane, T. R. and Levinson, D. A. (2000). Locomotion of snakes: A mechanical 'explanation'. *Int. Journ. of Solids and Structures*, 37(41):5829–5837.

Kanso, E., Marsden, J. E., Rowley, C. W., and Melli-Huber, J. B. (2005). Locomotion of articulated bodies in a perfect fluid. *Journ. of Nonlinear Science*, 15(4):225–289.

Keladisi, E., Jesmani, M., Pettersen, K. Y., and Gravdahl, J. T. (2016a). Multi-objective optimization for efficient motion of underwater snake robots. *Artificial Life and Robotics*, 21(4):411–422.

Keladisi, E., Liljebäck, P., Pettersen, K. Y., and Gravdahl, J. T. (2015a). Experimental investigation of efficient locomotion of underwater snake robots for lateral undulation and eel-like motion patterns. *Robotics and Biomimetics*, 2(1):1–27.

Keladisi, E., Pettersen, K. Y., and Gravdahl, J. T. (2014a). A control-oriented model of underwater snake robots. In *Proc. IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, pages 753–760, Bali, Indonesia.

Keladisi, E., Pettersen, K. Y., and Gravdahl, J. T. (2014b). Stability analysis of underwater snake robot locomotion based on averaging theory. In *Proc. IEEE Int. Conf. on Robotics and Biomimetics (ROBIO)*, pages 574–581, Bali, Indonesia.

Keladisi, E., Pettersen, K. Y., and Gravdahl, J. T. (2015b). Energy efficiency of underwater snake robot locomotion. In *Proc. 23rd Mediterranean Conf. on Control Automation (MED)*, pages 1124–1131, Torremolinos, Spain.

Keladisi, E., Pettersen, K. Y., Gravdahl, J. T., and Liljebäck, P. (2014c). Modeling of underwater snake robots. In *Proc. IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 4540–4547, Hong Kong, China.

Keladisi, E., Pettersen, K. Y., Liljebäck, P., and Gravdahl, J. T. (2014d). Integral line-of-sight for path-following of underwater snake robots. In *Proc. IEEE Multi-Conf. on Systems and Control*, pages 1078–1085, Juan Les Antibes, France.

Keladisi, E., Pettersen, K. Y., Liljebäck, P., and Gravdahl, J. T. (2016b). Locomotion efficiency of underwater snake robots with thrusters. In *Proc. IEEE Int. Symp. on Safety, Security, and Rescue Robotics (SSRR)*, pages 174–181, Lausanne, Switzerland.

Khalil, W., Gallot, G., and Boyer, F. (2007). Dynamic modeling and simulation of a 3-d serial eel-like robot. *IEEE Trans. on Systems, Man, and Cybernetics*, 37(6):1259–1268.

Kim, J., Yang, I., and Lee, D. (2013). Daisy chain method for control allocation based fault-tolerant control. *IEMEK Journ. of Embedded Systems and Applications*, 8(5):265–272.

Krishnaprasad, P. S. and Tsakiris, D. P. (1994). G-snakes: Nonholonomic kinematic chains on lie groups. In *Proc. 33rd IEEE Conf. Decision and Control*, volume 3, pages 2955–2960, Lake Buena Vista, FL USA.

Kuwada, A., Wakimoto, S., Suzumori, K., and Adomi, Y. (2008). Automatic pipe negotiation control for snake-like robot. In *Proc. IEEE/ASME Int. Conf. on Advanced Intelligent Mechatronics*, pages 558–563, Xi'an, China.

Lampetra project, SSSA. The BioRobotics Institute, Scuola Superiore Sant'Anna (SSSA), `http://sssa.bioroboticsinstitute.it/projects/LAMPETRA`, Viewed: 05.06.2018.

Li, B., Yu, S., Ma, S., and Wang, Y. (2011). An amphibious snake-like robot with novel gaits on ground and in water. In *Proc. IASTED Int. Conf. on Intelligent Systems and Control (ISC)*, pages 100–105, Cambridge, United Kingdom.

Li, J. and Shan, J. (2008). Passivity control of underactuated snake-like robots. In *Proc. 7th World Congress in Intelligent Control and Automation*, pages 485–490.

Liljebäck, P., Pettersen, K. Y., Stavdahl, Ø., and Gravdahl, J. T. (2010a). Fundamental properties of snake robot locomotion. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2876–2883, Taipei, Taiwan.

Liljebäck, P., Pettersen, K. Y., Stavdahl, Ø., and Gravdahl, J. T. (2010b). A simplified model of planar snake robot locomotion. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2868–2875, Taipei, Taiwan.

Liljebäck, P., Pettersen, K. Y., Stavdahl, Ø., and Gravdahl, J. T. (2011). Controllability and stability analysis of planar snake robot locomotion. *IEEE Trans. on Automatic Control*, 56(6):1365–1380.

Liljebäck, P., Pettersen, K. Y., Stavdahl, Ø., and Gravdahl, J. T. (2012). A review on modelling, implementation, and control of snake robots. *Robotics and Autonomous Systems*, 60(1):29–40.

Liljebäck, P., Pettersen, K. Y., Stavdahl, Ø., and Gravdahl, J. T. (2013). *Snake Robots: Modelling, Mechatronics, and Control*. Springer, London.

Liljebäck, P., Stavdahl, Ø., and Pettersen, K. Y. (2008). Modular pneumatic snake robot: 3d modelling, implementation and control. *Modeling, Identification and Control (MIC)*, 29(1):21–28.

Liljebäck, P., Stavdahl, Ø., Pettersen, K. Y., and Gravdahl, J. T. (2014). Mamba - a waterproof snake robot with tactile sensing. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 294–301, Chicago, IL USA.

Lindfors, I. (1993). Thrust allocation method for the dynamic positioning system. In *10th Int. Ship Control Systems Symposium*, pages 3.93–3.106, Ottawa, Canada.

Ma, S. (1999). Analysis of snake movement forms for realiztion of snake-like robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, volume 4, pages 3007–3013, Detroit, MI USA.

Ma, S. (2001). Analysis of creeping locomotion of a snake-like robot. *Advanced Robotics*, 15(2):205–224.

Ma, S., Araya, H., and Li, L. (2001). Development of creeping snake-robot. In *Proc. IEEE Int. Symp. on Computational Intelligence in Robotics and Automation*, pages 77–82.

Ma, S., Ohmameuda, Y., and Inoue, K. (2004). Dynamic analysis of 3-dimensional snake robots. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 767–772, Sendal Japan.

Ma, S., Ohmameuda, Y., Inoue, K., and Li, B. (2003). Control of 3-dimensional snake-like robot. In *Proc. IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 2067–2072.

Ma, S. and Tadokoro, N. (2006). Analysis of creeping locomotion of a snake-like robot on a slope. *Autonomous Robots*, 20:15–23.

Math Works Inc. `https://se.mathworks.com/`, Viewed: 13.06.2018.

Matsuno, F. and Mogi, K. (2000). Redundancy controllable system and control of snake robots based on kinematic model. In *Proc. IEEE Conf. on Decision and Control*, volume 5, pages 4791–4796.

Matsuno, F. and Sato, H. (2005). Trajectory tracking control of snake robots based on dynamic model. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 3029–3034.

McIsaac, K. A. and Ostrowski, J. P. (1999). A geometric approach to anguilliform locomotion: Modelling of an underwater eel robot. In *Proc. IEEE Int. Conf. in Robotics and Automation*, pages 2843–2848, Detroit, MI USA.

McIsaac, K. A. and Ostrowski, J. P. (2000). Motion planning for dynamic eel-like robots. In *Proc. IEEE Int. Conf. in Robotics and Automation*, pages 1695–1700, San Francisco, CA USA.

McIsaac, K. A. and Ostrowski, J. P. (2002). Experiments in closed-loop control for an undewater eel-like robot. In *Proc. IEEE Int. Conf. on Robotics & Automation*, pages 750–755, Washington DC, USA.

McIsaac, K. A. and Ostrowski, J. P. (2003). Motion planning for anguilliform locomotion. *IEEE Trans. on Robotics and Automation*, 19(4):637–652.

McMillan, S., Orin, D. E., and McGhee, R. B. (1995). Efficient dynamic simulation of an underwater vehicle with a robotic manipulator. *IEEE Trans. on System, Man, and Cybernetics*, 25(8):1194–1206.

Mehta, V., Brennan, S., and Gandhi, F. (2008). Experimentally verified optimal serpentine gait and hyperredundancy of a rigid-link snake robot. *IEEE Trans. on Robotics*, 24(2):348–360.

Moon, B. R. and Gans, C. (1998). Kinematics, muscular activity and propulsion in gopher snakes. *Journal of Experimental Biology*, 201:2669–2684.

Mori, M. and Hirose, S. (2002). Three-dimensional serpentine motion and lateral rolling by active cord mechanism ACM-R3. In *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, pages 829–834.

Nilsson, M. (1998). Snake robot - free climbing. *IEEE Control Systems Magazine*, 18(1):21–26.

Nilsson, M. (2004). Serpentine locomotion on surface with uniform friction. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 1751–1755, Sendal, Japan.

Nocedal, J. and Wright, S. J. (2006). *Numerical Optimization*. Springer, New York, USA, 2nd edition.

Ohashi, T., Yamada, H., and Hirose, S. (2010). Loop forming snake-like robot ACM-R7 and its serpenoid oval control. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 413–418, Taipei, Taiwan.

Ohno, H. and Hirose, S. (2001). Design of slim slime robot and its gait of locomotion. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 707–715.

Ostrowski, J. and Burdick, J. (1998). The geometric mechanics of undulatory robotic locomotion. *Int. Journ. of Robotics Research*, 17(7):683–701.

Paradiso, J. A. (1991). Adaptable method of managing jets and aerosurfaces for aerospace vehicle control. *Journ. of Guidance, Control, and Dynamics*, 14(1):44–50.

Petersen, J. A. M. and Bodson, M. (2005). Interior-point algorithms for control allocation. *Journ. of Guidance, Control, and Dynamics*, 28(3):471–480.

Petersen, J. A. M. and Bodson, M. (2006). Constrained quadratic programming techniques for control allocation. *IEEE Trans. on Control Systems Technology*, 14(1):91–98.

Porez, M., Boyer, F., and Ijspeert, A. J. (2014). Improved lighthill fish swimming model for bio-inspired robots: Modeling, computational aspects and experimental comparison. *The Int. Journ. of Robotics Research*, 33(10):1322–1341.

Prautsch, P. and Mita, T. (1999). Control and analysis of the gait of snake robots. In *Proc. IEEE Int. Conf. on Control Applications*, pages 502–507, Kohala Coast, HI USA.

Saito, M., Fukaya, M., and Iwasaki, T. (2002). Serpentine locomotion with robotic snakes. *IEEE Control Systems Magazine*, 22(1):64–81.

Spong, M. W., Hutschinson, S., and Vidyasagar, M. (2006). *Robot Modeling and Control*. John Wiley & Sons, Inc., Hoboken, NJ USA, 1. edition.

Stafanini, C., Orofino, S., Manfredi, L., Mintchev, S., Marrazza, S., Assaf, T., Capantini, L., Sinibaldi, E., Grillner, S., Wallén, P., and Dario, P. (2012). A novel autonomous, bioinspired swimming robot developed by neuroscientists and bioengineers. *Bioinspiration & Biomimetics*, 7(2):025001, 8pp.

Sverdrup-Thygeson, J., Keladisi, E., Pettersen, K. Y., and Gravdahl, J. T. (2016a). A control framework for biologically inspired underwater swimming manipulators equipped with thrusters. In *Proc. 10th IFAC Conf. on Control Applications in Marine Systems (CAMS)*, pages 89–96, Trondheim, Norway.

Sverdrup-Thygeson, J., Keladisi, E., Pettersen, K. Y., and Gravdahl, J. T. (2016b). Modeling of underwater swimming manipulators. In *Proc. 10th IFAC Conf. on Control Applications in Marine Systems (CAMS)*, pages 81–88, Trondheim, Norway.

Sverdrup-Thygeson, J., Keladisi, E., Pettersen, K. Y., and Gravdahl, J. T. (2018). The underwater swimming manipulator - a bioinspired solution for subsea operations. *IEEE Journ. of Oceanic Engineering*, 43(2):402–417.

Sørdalen, O. J. (1997). Optimal thrust allocation for marine vessels. *Control Engineering Practice*, 5(9):1223–1231.

Takayama, T. and Hirose, S. (2002). Amphibious 3d active cord mechanism "HELIX" with helical swimming motion. In *Proc.IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 775–780, Lausanne, Switzerland.

Tanaka, M. and Matsuno, F. (2008). Modeling and control of a snake robot with switching constraints. In *SICE Annual Conference*.

Togawa, K., Mori, M., and Hirose, S. (2000). Study on three-dimentional active chord mechanism: Developement of ACM-R2. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, volume 3, pages 2242–2247.

Transeth, A. A., Leine, R. I., Glocker, C., and Pettersen, K. Y. (2008). 3-d snake robot motion: Nonsmooth modeling, simulations, and experiments. *IEEE Trans. on Robotics*, 24(2):361–376.

Transeth, A. A., Pettersen, K. Y., and Liljebäck, P. (2009). A survey on snake robot modeling and locomotion. *Reliability Engineering and System Safety*, 93:1208–1217.

Ute, J. and Ono, K. (2002). Fast and efficient locomotion of a snake robot based on self-excitation principle. In *Proc. 7th Int. Workshop on Advanced Motion Control*, pages 532–539.

Virnig, J. and Bodden, D. (1994). Multivariable control allocation and control law conditioning when control effectors limit. In *Proc. AIAA Guidance, Navigation and Control Conf.*, pages 572–582, Scottsdale, AZ USA.

Wiens, A. J. and Nahon, M. (2012). Optimally efficient swimming in hyper-redundant mechanisms: control, design, and energy recovery. *Bioinspiration & Biomimetics*, 7(4):046016, 13pp.

Wiriyacharoensunthorn, P. and Laowattana, S. (2002). Analysis and design of a multi-link mobile robot (serpentine). In *Proc. IEEE Int. Conf. on Robotics, Intelligent Systems and Signal Processing*, pages 694–699.

World Geodetic System (1984). Its Definition and Relationships with Local Geoedtic Systems. DMA TR 8350.2, 2nd ed.,Defense Mapping Agency, Fairfax, VA.

Worst, R. and Linnemann, R. (1996). Construction and operation of a snake-like robot. In *Proc. IEEE Int. Joint Symp. on Intelligence and Systems*, pages 164–169, Rockville, MD USA.

Wright, C., Johnson, A., Peck, A., McCord, Z., Naaktgeboren, A., Gianfortoni, P., Gonzalez-Rivero, M., Hatton, R., and Choset, H. (2007). Design of a modular snake robot. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2609–2614, San Diego, CA USA.

Ye, C., Ma, S., Li, B., and Wang, Y. (2004). Locomotion control of a novel snake-like robot. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, volume 1, pages 829–834.

Yim, M., Duff, D., and Roufas, K. (2002). Walk on the wild side. *IEEE Robotics & Automation Magazine*, 9(4):49–53.

Yu, S., Ma, S., Li, B., and Wang, Y. (2009). An amphibious snake-like robot: Design and motion experiments on ground and in water. In *Proc. IEEE Int. Conf. on Information and Automation*, pages 500–505.

Zuo, Z., Wang, Z., Li, B., and Ma, S. (2008). Serpentine locomotion of a snake-like robot in water environment. In *IEEE Int. Conf. on Robotics and Biomimetics*, pages 25–30, Bangkok, Thailand.

Appendix A

Additional Simulation Results

A.1 CASE 1 - Unconstrained Thrust Allocation

A.1.1 Planar Motion

Standard Damped Inverse

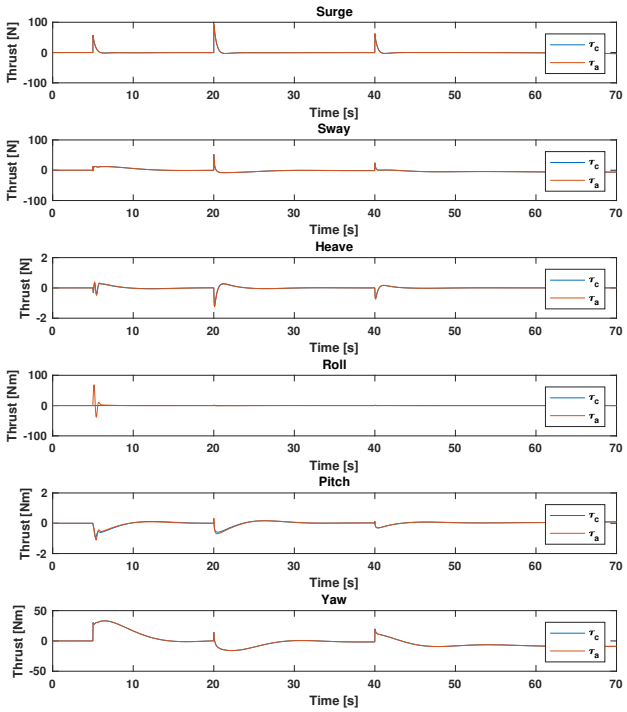


Figure A.1: Commanded thrust τ_c and actual thrust τ_a for unconstrained thrust allocation for planar motion using standard damped inverse

Linear Programming

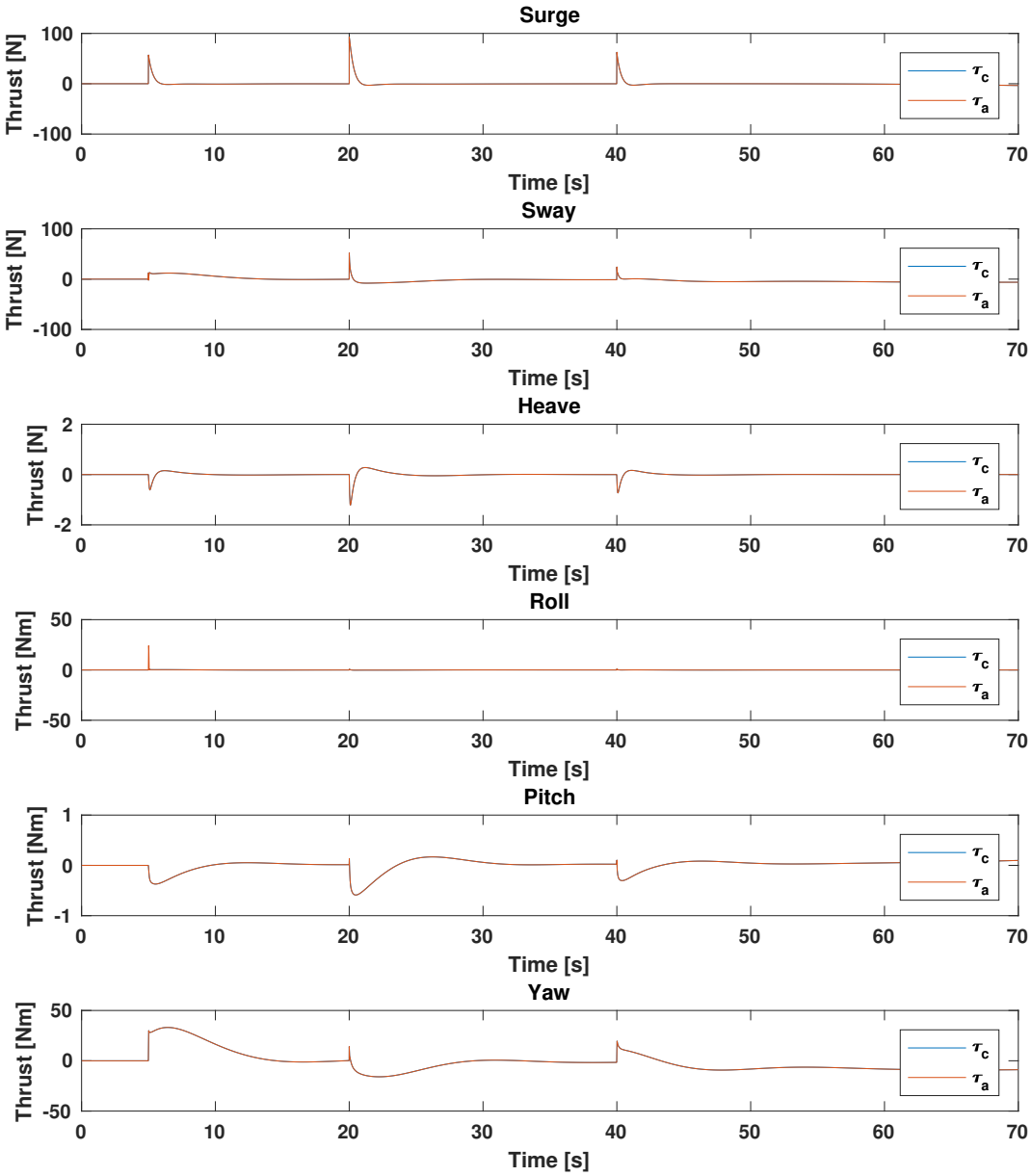


Figure A.2: Commanded thrust τ_c and actual thrust τ_a for unconstrained thrust allocation for planar motion using linear programming

Quadratic Programming

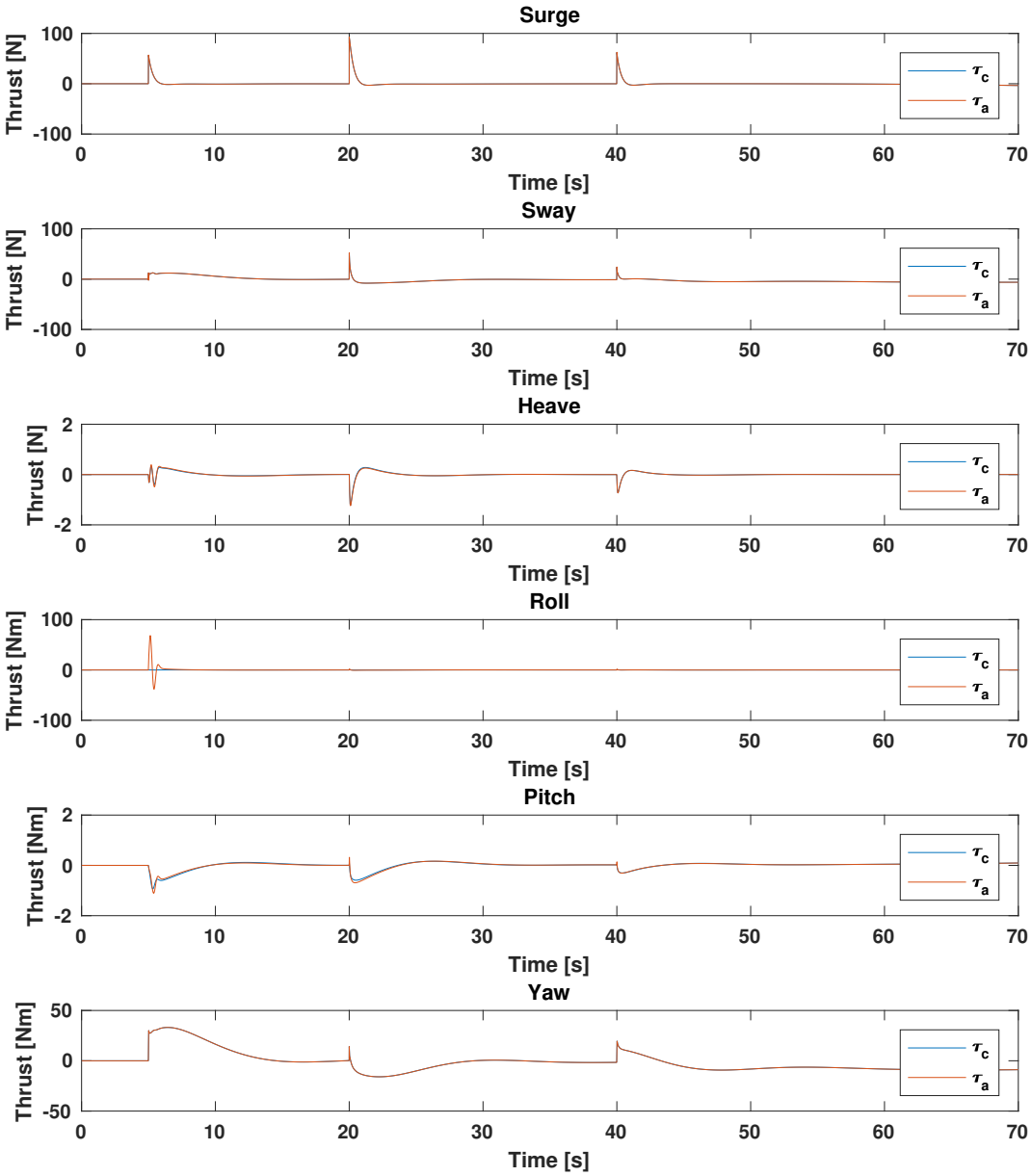


Figure A.3: Commanded thrust τ_c and actual thrust τ_a for unconstrained thrust allocation for planar motion using quadratic programming

A.1.2 Three-dimensional motion

Standard Damped Inverse

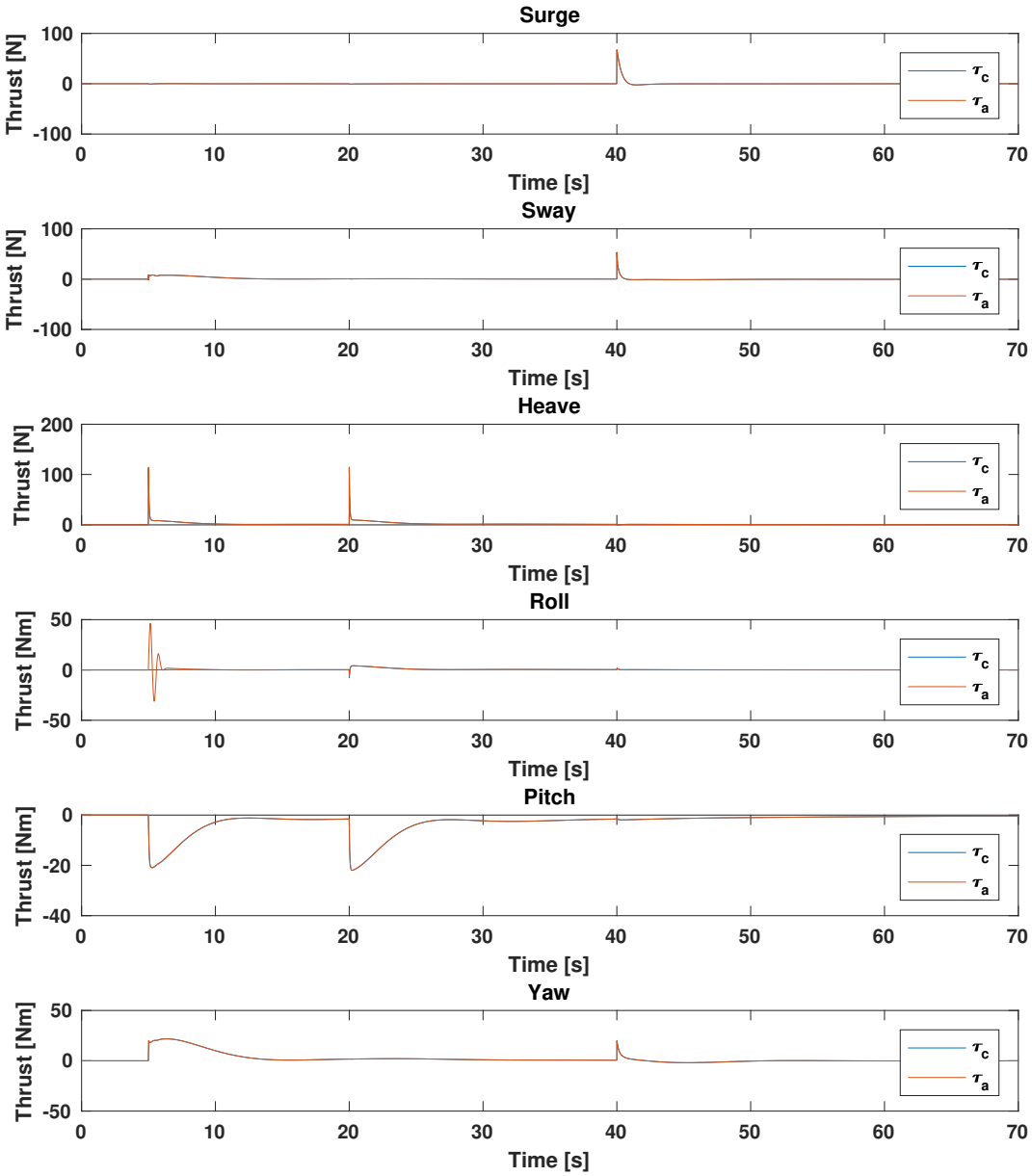


Figure A.4: Commanded thrust τ_c and actual thrust τ_a for unconstrained thrust allocation for three-dimensional using standard damped inverse

Linear Programming

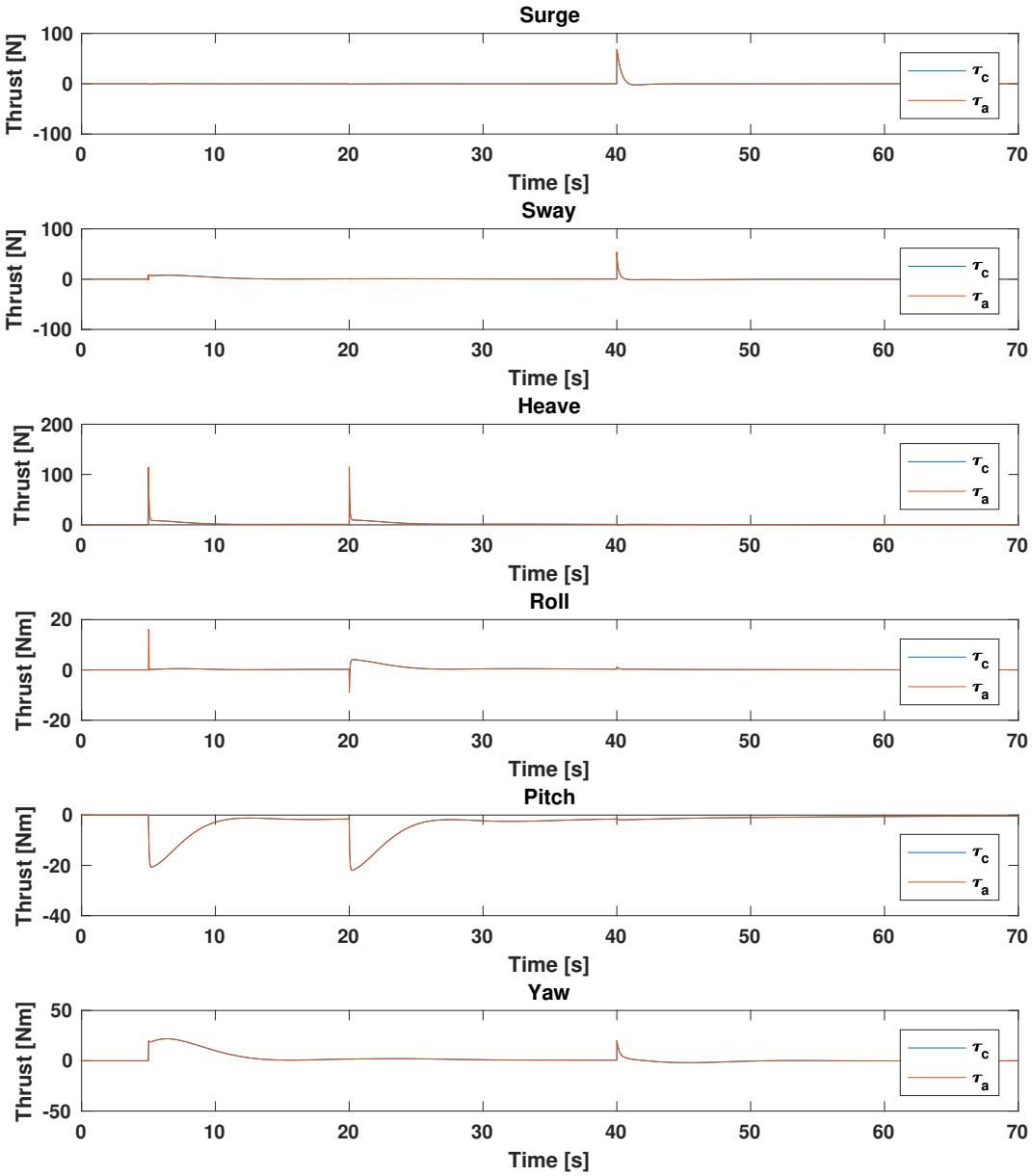


Figure A.5: Commanded thrust τ_c and actual thrust τ_a for unconstrained thrust allocation for three-dimensional motion using linear programming

Quadratic Programming

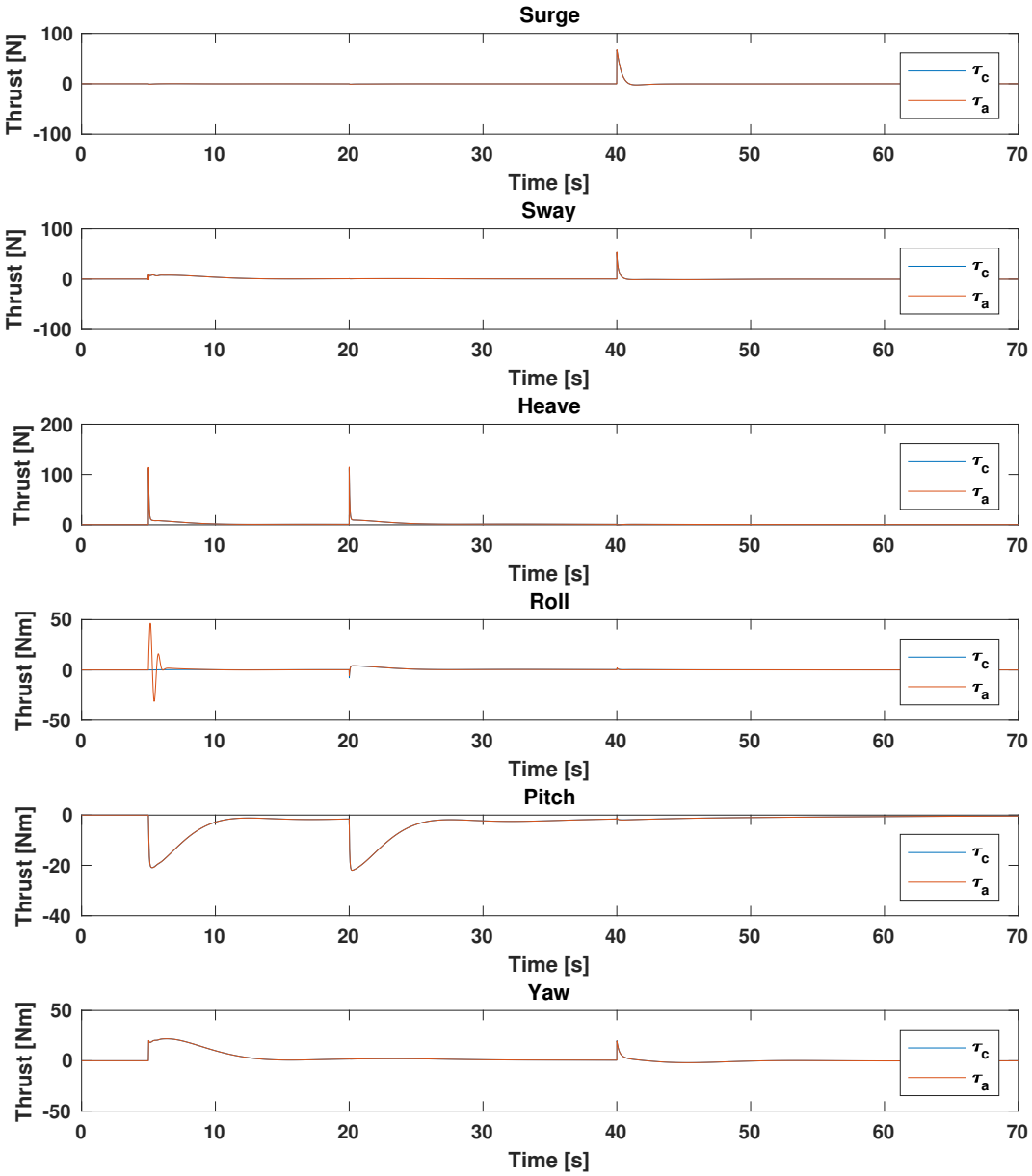


Figure A.6: Commanded thrust τ_c and actual thrust τ_a for unconstrained thrust allocation for three-dimensional motion using linear programming

A.2 CASE 2 - Constrained Thrust Allocation

A.2.1 Planar Motion

Redistributed Pseudo-Inverse

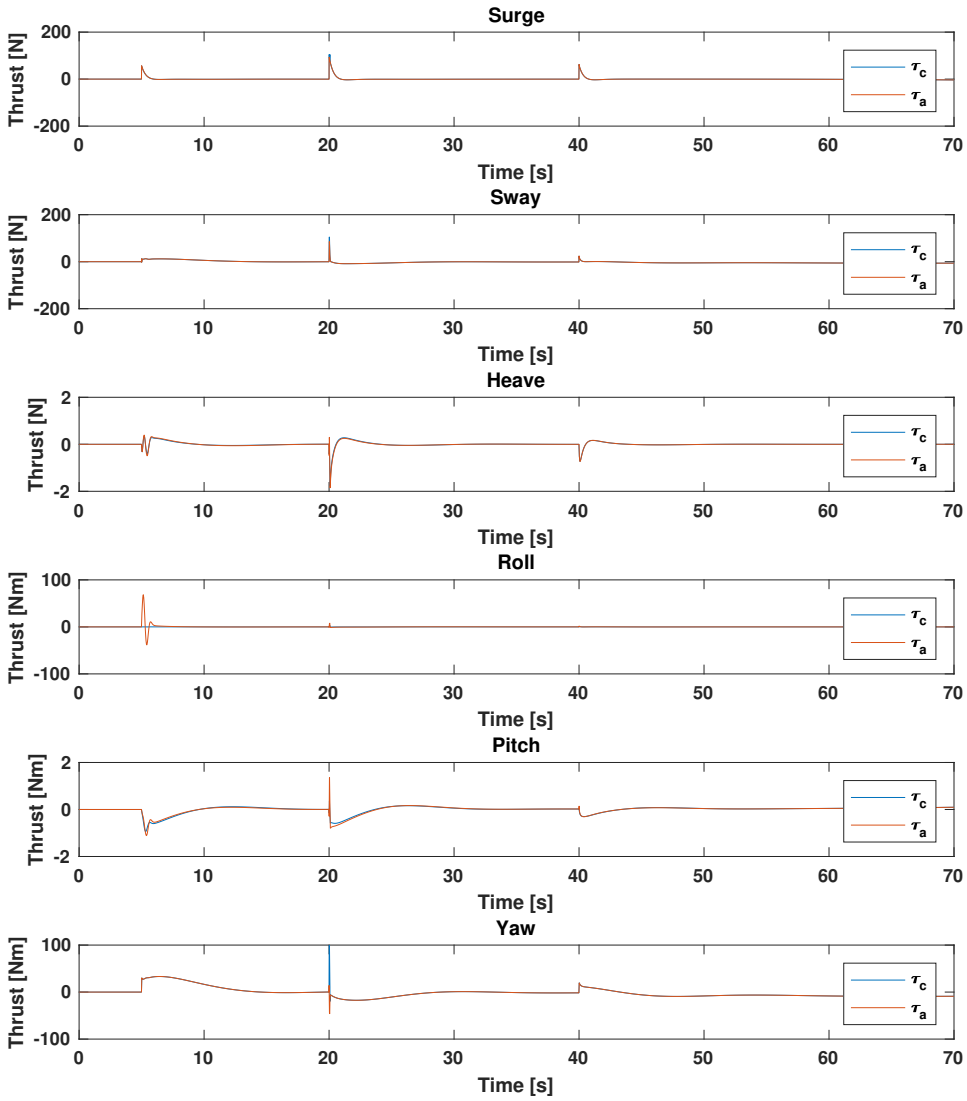


Figure A.7: Commanded thrust τ_c and actual thrust τ_a for constrained thrust allocation for planar motion using redistributed pseudo-inverse

Linear Programming

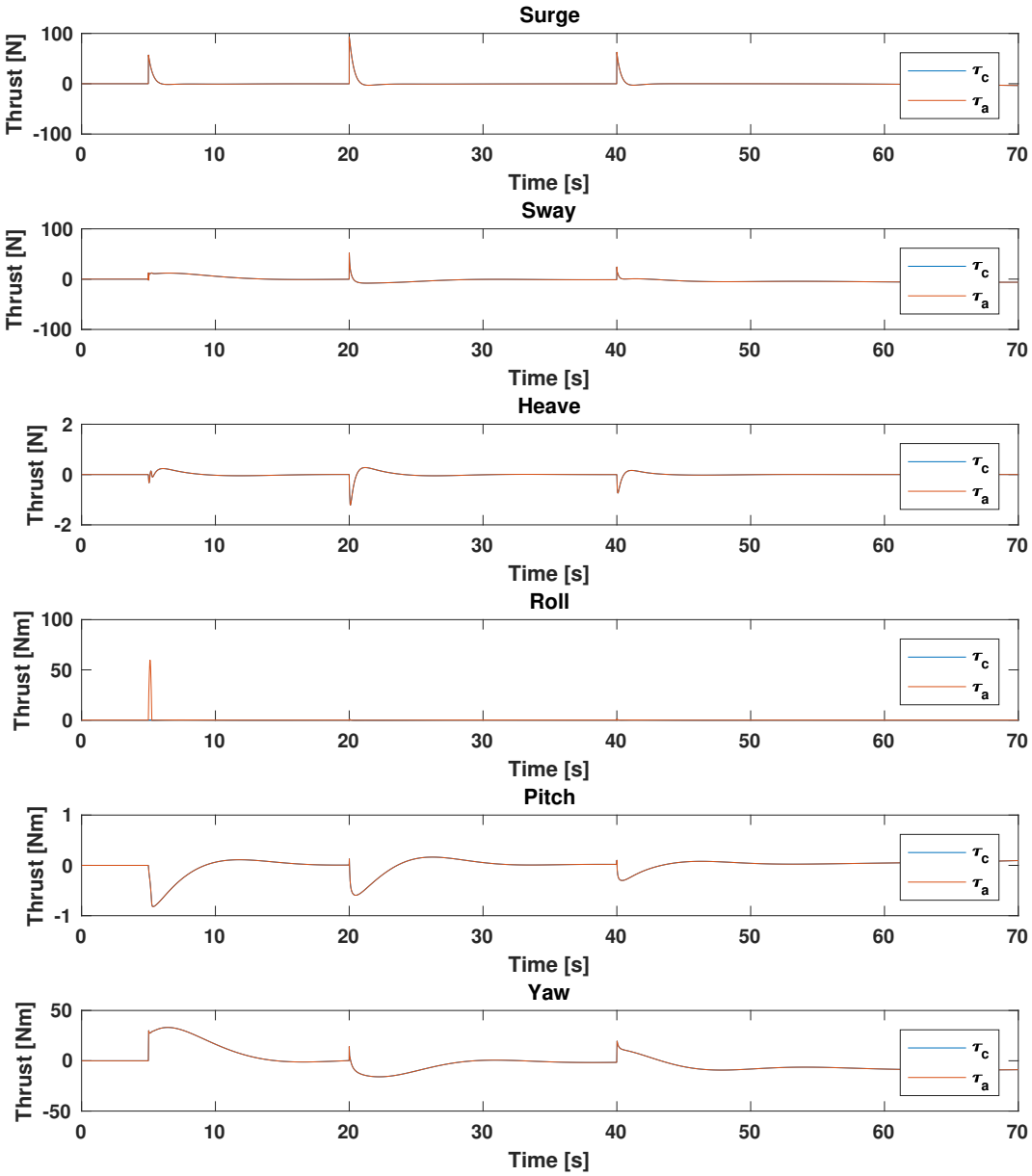


Figure A.8: Commanded thrust τ_c and actual thrust τ_a for constrained thrust allocation for planar motion using linear programming

Quadratic Programming

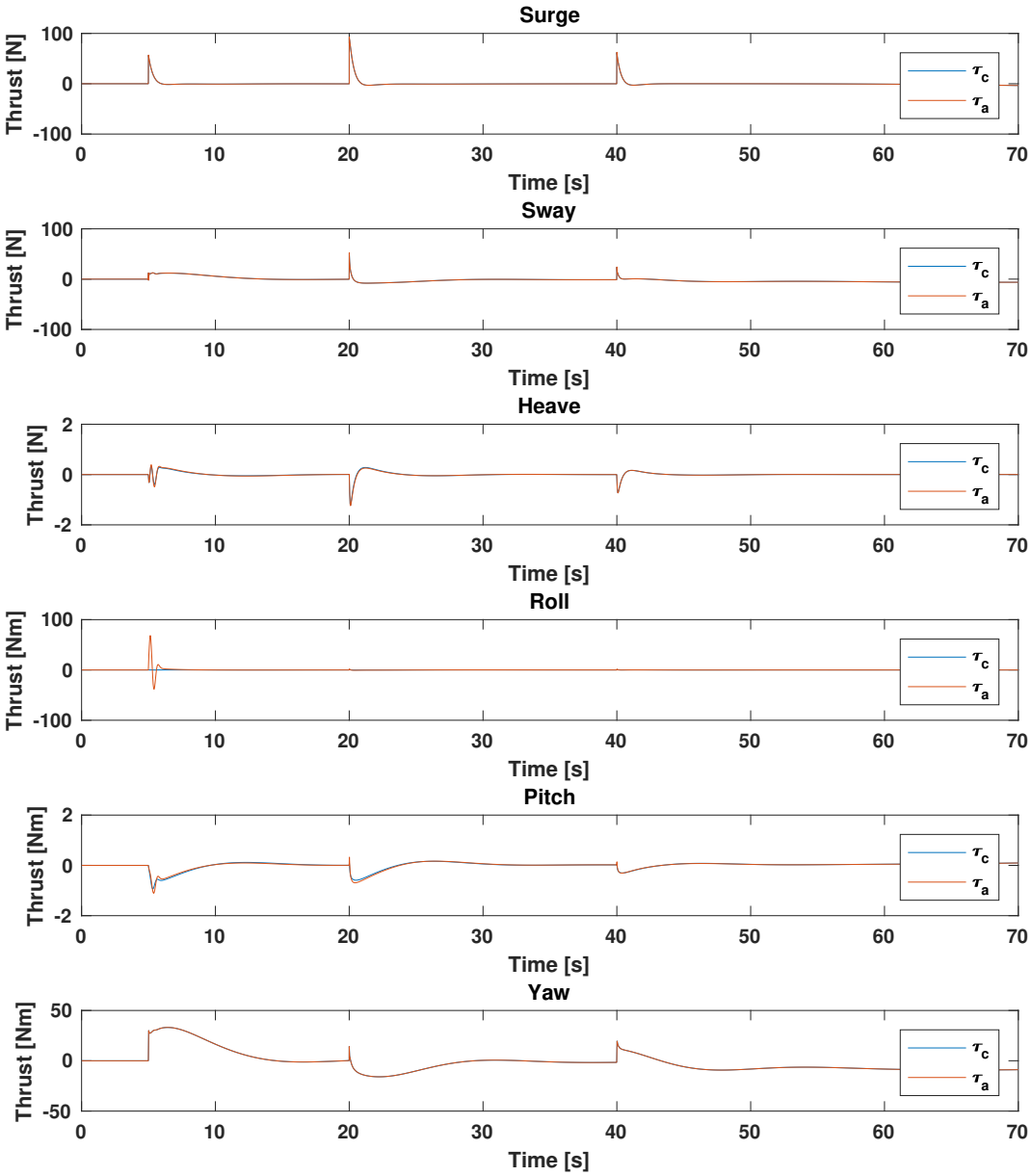


Figure A.9: Commanded thrust τ_c and actual thrust τ_a for constrained thrust allocation for planar motion using quadratic programming

A.2.2 Three-dimensional motion

Redistributed Pseudo-Inverse

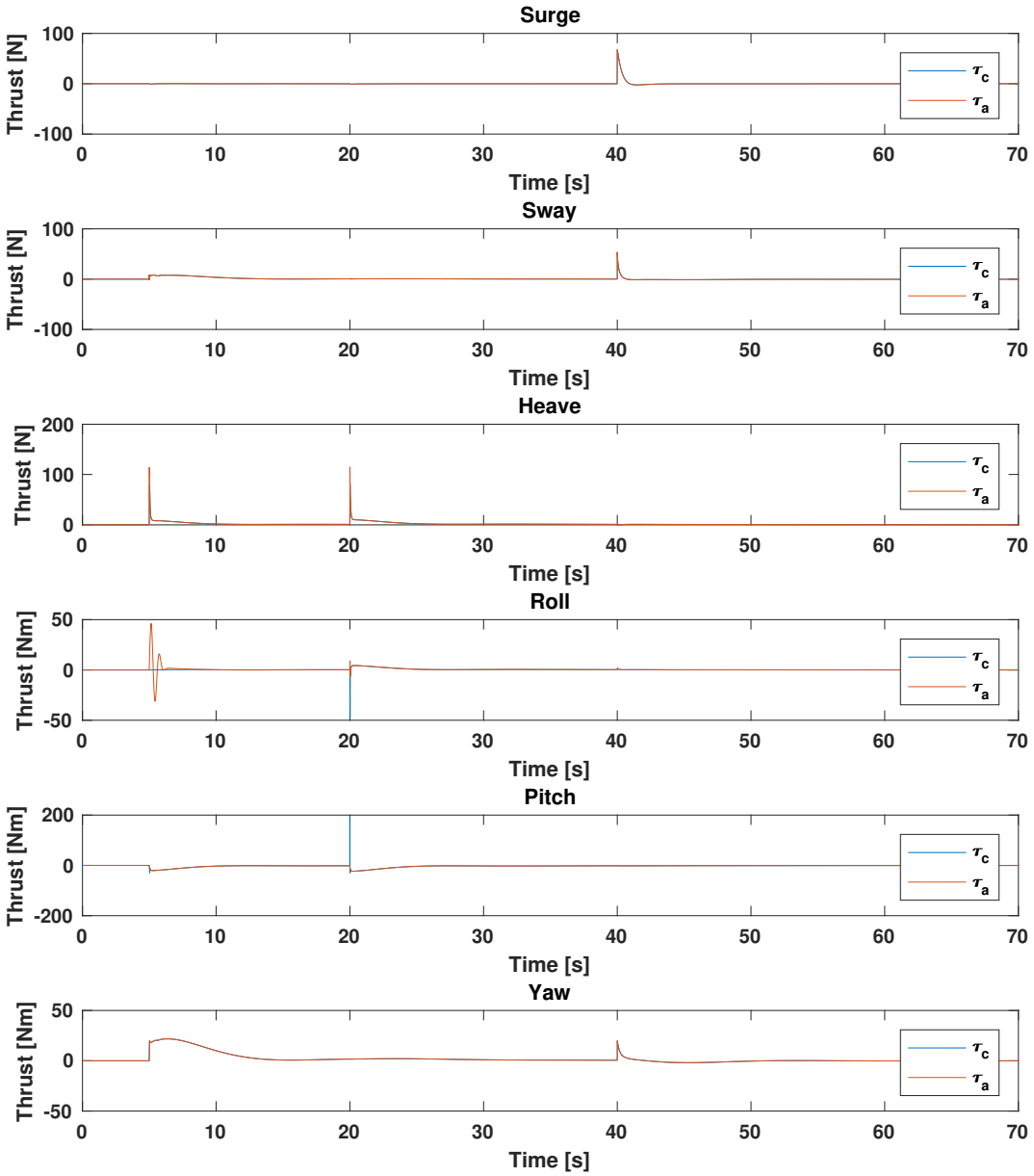


Figure A.10: Commanded thrust τ_c and actual thrust τ_a for constrained thrust allocation for three-dimensional using redistributed pseudo-inverse

Linear Programming

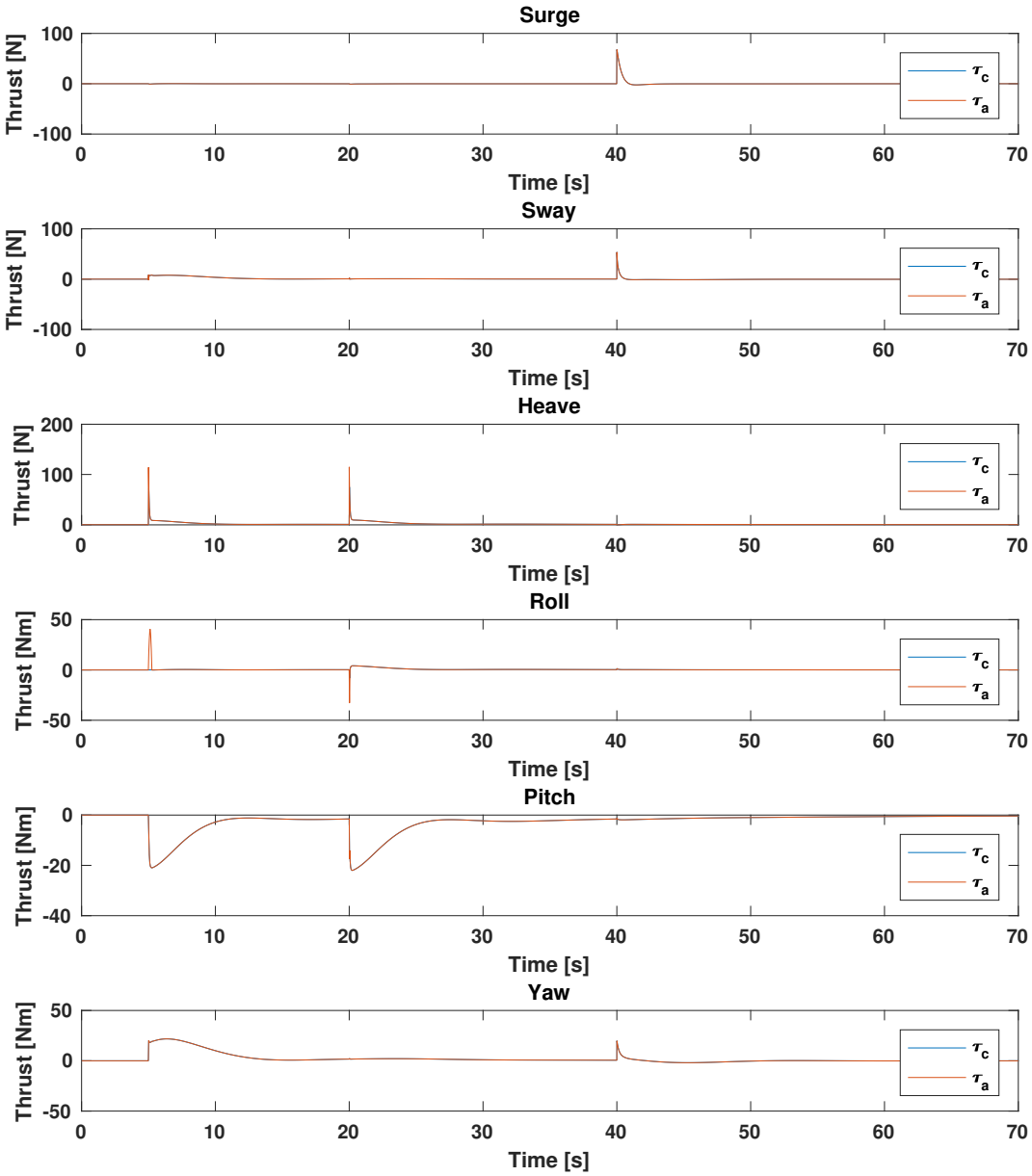


Figure A.11: Commanded thrust τ_c and actual thrust τ_a for constrained thrust allocation for three-dimensional motion using linear programming

Quadratic Programming

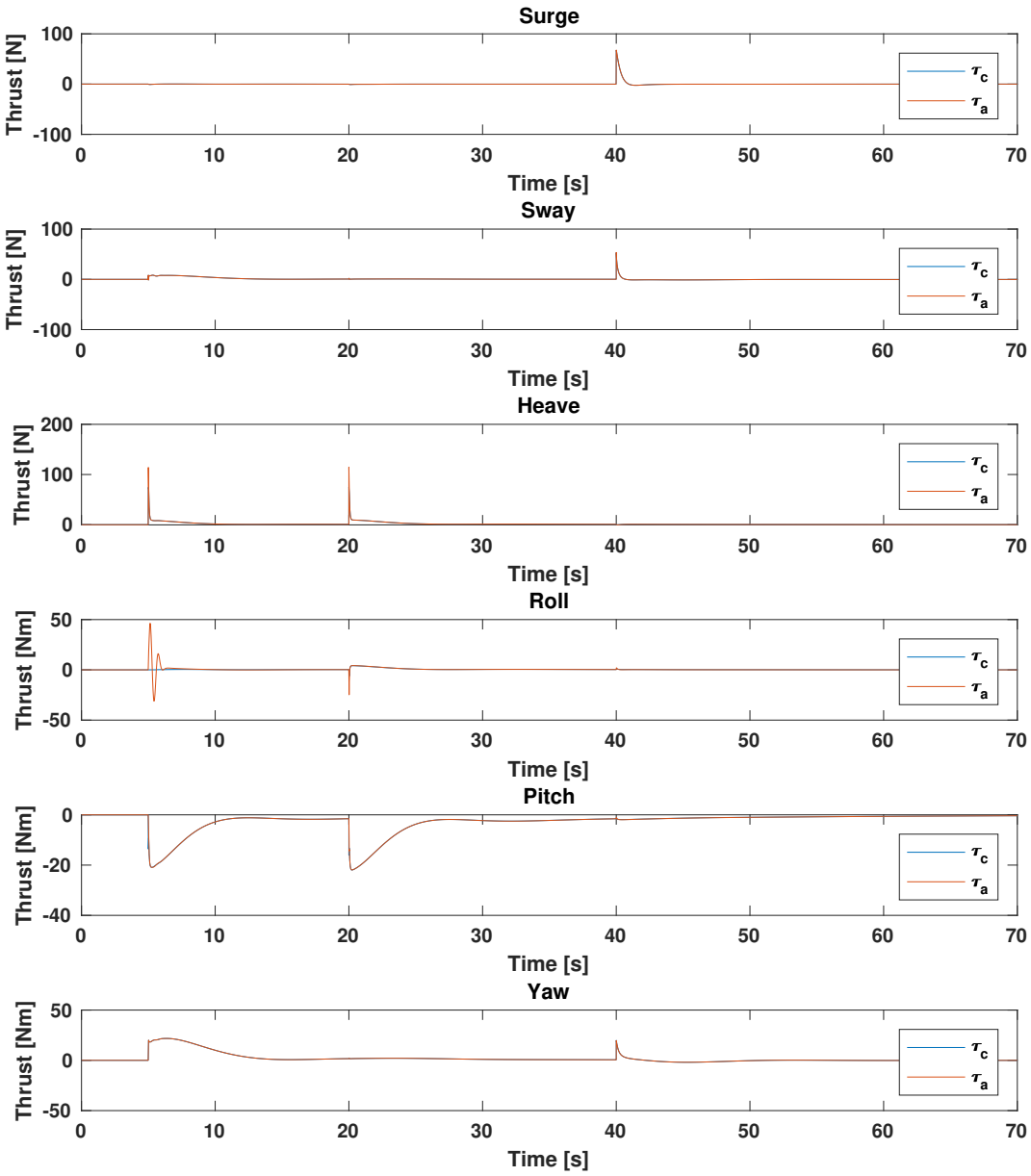


Figure A.12: Commanded thrust τ_c and actual thrust τ_a for constrained thrust allocation for three-dimensional motion using quadratic programming

Appendix B

MATLAB Code

B.1 Linear Programming Algorithm

```
function [f_thr, s] = LPmixed(tau_c, TCM, snake)
%==========================================================================
% LPmixed - This function calculates thruster forces using a linear
% programming algorithm. The states to be optimized are x = [s+ s- u+ u-]'
%
% Syntax:  [f_thr,s] = LPmixed(tau_c,TCM,snake)
%
% Inputs:
%     tau_c  - Commanded thrust vector
%     TCM    - Thrust configuration matrix
%     snake  - Snake robot model properties
%
% Outputs:
%     f_thr  - Thruster forces
%     s      - Slack variable
%
% Author: Siri Bjoerkedal OEvregaard
% Date:   23.06.18
%==========================================================================

% Total number of thrusters
N = snake.n_thruster_tot;

```

```
24 % Thruster saturation limit
25 U_max = 1000;        % 1000 for CASE 1, 40 for CASE 2 (constrained)
26 U_min = -1000;       % -1000 for CASE 1, -40 for CASE 2 (constrained)
27
28 % Preferred thrust vector
29 u_p = zeros(N,1);
30
31 % Slack variable limit
32 s_max = norm(TCM*u_p-tau_c,1);
33
34 % Weight parameter
35 e = 1e-5;
36
37 % Weight vectors
38 w = ones(6,1);        % Error weight
39 q = ones(N,1)*e;      % Thrust weight
40
41 I_s = eye(6);
42
43 %==================================
44
45 % Defining linear program model
46 f = [w; w; q; q];
47
48 A_eq = [I_s -I_s -TCM TCM];
49
50 b_eq = TCM*u_p - tau_c;
51
52 lb = zeros(6*2+N*2,1);
53 ub = [ones(6,1)*s_max; ones(6,1)*s_max; U_max-u_p; u_p-U_min];
54
55 % Deactivating writing to command window
56 options = optimoptions('linprog','Display','off');
57
58 % Solving linear program
59 sol = linprog(f,[],[],A_eq,b_eq,lb,ub,options);
60
```

```
61 % Extracting output variables
62 s_pluss = sol(1:6);
63 s_minus = sol(7:12);
64 u_pluss = sol(13:12+N);
65 u_minus = sol(13+N:12+2*N);
66
67 s = s_pluss - s_minus;
68
69 f_thr = u_pluss - u_minus + u_p;
```

B.2 Quadratic Programming Algorithm

```
function [f_thr, s] = QP(tau_c, TCM, snake)
%=========================================================================
% QP - This function calculates thruster forces using a quadratic
% programming algorithm. The states to be optimized are x = [u s]'.
%
% Syntax:  [f_thr,s] = QP(tau_c,TCM,snake)
%
% Inputs:
%     tau_c  - Commanded thrust vector
%     TCM    - Thrust configuration matrix
%     snake  - Snake robot model properties
%
% Outputs:
%     f_thr  - Thruster forces
%     s      - Slack variable
%
% Author: Siri Bjoerkedal OEvregaard
% Date:   23.06.18
%=========================================================================

% Total number of thrusters
N = snake.n_thruster_tot;

% Thruster saturation limit
u_max = 40;          % 1000 for CASE 1, 40 for CASE 2 (constrained)
u_min = -40;         % -1000 for CASE 1, -40 for CASE 2 (constrained)

% Weight parameter
Q = inv(0.02);

% Weight vectors
w = ones(6,1)*Q;          %Error weights
q = ones(N,1)*1;          %Thrust weights
weight = [q; w];

```

```
36 I_u = eye(N);
37 I_s = eye(6);
38
39 %==================================
40
41 % Defining quadratic program model
42 H = 2*diag(weight);
43
44 f = zeros(N+6,1);
45
46 A = [-I_u zeros(N,6);
47      I_u zeros(N,6)];
48 b = [ones(N,1)*-u_min;
49      ones(N,1)*u_max];
50
51 A_eq = [TCM -I_s];
52 b_eq = tau_c;
53
54 % Deactivating writing to command window
55 options = optimoptions('quadprog','Display','off');
56
57 % Solving quadratic program
58 sol = quadprog(H,f,A,b,A_eq,b_eq,[],[],[],options);
59
60 % Extracting output variables
61 u = sol(1:N);
62 s = sol(N+1:N+6);
63
64 f_thr = u;
```

B.3 Redistributed Pseudo-Inverse Algorithm

```
function [f_thr] = RPI(TCM, Q_inv, tau_c)
%#codegen
%=========================================================================
% RPI - This function calculates thruster forces using a redistributed
% pseudo-inverse algorithm.
%
% Syntax:  [f_thr] = RPI(TCM,Q_inv,tau_c)
%
% Inputs:
%    TCM    - Thrust configuration matrix
%    Q_inv  - Damping coefficient
%    tau_c  - Commanded thrust vector
%
% Outputs:
%    f_thr  - Thruster forces
%
% Author: Siri Bjoerkedal OEvregaard
% Date:   23.06.18
%=========================================================================

% Declaring variable size variables
coder.varsize('f_thr_curr','TCM_curr','count_u_vector','f_th')

% Thruster saturation limit
u_max = 40;          % 1000 for CASE 1, 40 for CASE 2 (constrained)
u_min = -40;         % -1000 for CASE 1, -40 for CASE 2 (constrained)


%Calculating first standard inverse
TCM_inv = TCM'/(TCM*TCM' + Q_inv);
TCM_inv(abs(TCM_inv) < 1e-15) = 0;

%Calculating first thrust vector
f_th_1 = TCM_inv * tau_c;

```

```
% Declaring vector counting unsaturated elements
count_u_vector = zeros(length(f_th_1),1);
for i = 1:length(f_th_1)
    count_u_vector(i) = i;
end

% Defining auxiliary parameters
TCM_curr = TCM;
f_thr_sat = zeros(length(f_th_1),1);

f_th = f_th_1;

while max(f_th) > u_max || min(f_th) < u_min

    f_thr_curr = f_th;
    count = 0;      % Counting saturated elements in current thrust vector

    % Searching thrust vector for saturated elements
    for k = 1:length(f_th)

        if f_th(k) >= u_max

            %Setting value equal to saturation value
            f_th(k) = u_max;
            f_thr_sat(count_u_vector(k-count)) = u_max;

            % Defining new size of thrust vector and TCM matrix
            f_thr_new = zeros(length(f_thr_curr)-1,1);
            TCM_new = zeros(length(tau_c),length(f_thr_curr)-1);

            % Reducing thrust vector and TCM matrix  and TCM
            for i = 1:k-count-1
                f_thr_new(i) = f_thr_curr(i);
                TCM_new(:,i) = TCM_curr(:,i);
            end

            for i = k-count+1:length(f_thr_curr)
```

```
73                  f_thr_new(i-1) = f_thr_curr(i);
74                  TCM_new(:,i-1) = TCM_curr(:,i);
75              end
76
77              % Updating auxiliary variables and count variables
78              f_thr_curr = f_thr_new;
79              TCM_curr = TCM_new;
80              count_u_vector(k-count) = [];
81              count = count+1;
82
83          elseif f_th(k) <= u_min
84
85              %Setting value equal to saturation value
86              f_th(k) = u_min;
87              f_thr_sat(count_u_vector(k-count)) = u_min;
88
89              % Defining new size of thrust vector and TCM matrix
90              f_thr_new = zeros(length(f_thr_curr)-1,1);
91              TCM_new = zeros(length(tau_c),length(f_thr_curr)-1);
92
93              %Reducing f_thr and TCM
94              for i = 1:k-count-1
95                  f_thr_new(i) = f_thr_curr(i);
96                  TCM_new(:,i) = TCM_curr(:,i);
97              end
98
99              for i = k-count+1:length(f_thr_curr)
100                 f_thr_new(i-1) = f_thr_curr(i);
101                 TCM_new(:,i-1) = TCM_curr(:,i);
102             end
103
104             % Updating auxiliary variables and count variables
105             f_thr_curr = f_thr_new;
106             TCM_curr = TCM_new;
107             count_u_vector(k-count) = [];
108             count = count+1;
109
```

```
110             else
111                 f_th(k) = f_th(k);
112                 f_thr_sat(count_u_vector(k-count)) = 0;
113                 f_thr_curr(k-count) = f_th(k);
114             end
115         end
116
117         TCM = TCM_curr;
118
119         % Calculating new standard inverse
120         TCM_inv = TCM' / (TCM*TCM' + Q_inv);
121         TCM_inv(abs(TCM_inv) < 1e-15) = 0;
122
123         % Calculating new thrust vector
124         f_th = TCM_inv * tau_c;
125
126         % Checking if trust vector is empty
127         tf = isempty(f_th);
128         TF = double(tf);
129
130         % Exiting loop if thrust vector is empty
131         if TF == 1
132             break
133         end
134     end
135
136     % Reassembling thrust vector
137     for i = 1:length(count_u_vector)
138         f_thr_sat(count_u_vector(i)) = f_th(i);
139     end
140
141     f_thr = f_thr_sat;
```