Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Jonas Hopsdal

Adaptive Control of Underwater Snake Robot

Master's thesis in Marine Technology
Supervisor: Dong Trong Nguyen, Henrik Schmidt-Didlaukies
February 2019

NTNU
Norwegian University of
Science and Technology

Jonas Hopsdal

Adaptive Control of Underwater Snake Robot

Master's thesis in Marine Technology
Supervisor: Dong Trong Nguyen, Henrik Schmidt-Didlaukies
February 2019

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

NTNU Trondheim
Norwegian University of Science and Technology
Department of Marine Technology

MSC THESIS DESCRIPTION SHEET

Name of the candidate: Jonas Hopsdal

Field of study: Marine control engineering

Thesis title (Norwegian): Adaptiv kontroll for slangerobot under vann

Thesis title (English): Adaptive Control of Underwater Snake Robot

Background
Snake robots have been identified to work ideal for some subsea operations because of their ability to perform a wide range of operations such as inspections, maintenance and repair. As the snake robots will work under a range of scenarios and surroundings, the control system must be able to detect changes in surroundings and change the control law or control parameters accordingly. If achieved properly, the robot will be able to operate autonomously. To achieve this, an adaptive controller can be used. The adaptive controller must be stable and robust and at the same time adapt to the identified changes.

Work description
1. Perform a background and literature review to provide information and relevant references on:
   - Adaptive Control
   - Snake Robotics
   - Inverse Kinematics

   Write a list with abbreviations and definitions of terms, explaining relevant concepts related to the literature study and project assignment.

2. Study a mathematical model of the snake robot
3. Study a simulator for the snake robot
4. Implement a robust and stable adaptive control algorithm for a snake robot simulator
5. Test and validate the proposed adaptive controller by simulations of the snake robot in several scenarios.
6. Discuss the approach and results of simulation and conclude whether adaptive control could be useful for controlling a snake robot.

Specifications
The scope of work may prove to be larger than initially anticipated. By the approval from the supervisor, described topics may be deleted or reduced in extent without consequences with regard to grading.

The candidate shall present personal contribution to the resolution of problems within the scope of work. Theories and conclusions should be based on mathematical derivations and logic reasoning identifying the various steps in the deduction.

The report shall be organized in a logical structure to give a clear exposition of background, results, assessments, and conclusions. The text should be brief and to the point, with a clear language. Rigorous mathematical deductions and illustrating figures are preferred over lengthy textual descriptions. The report shall have font size 11 pts., and it is not expected to be longer than 60-80 A4 pages, from introduction to conclusion, unless otherwise agreed upon. It shall be written in English (preferably US) and contain the following elements: Title page, abstract, acknowledgements, thesis specification, list of symbols and acronyms, table of contents, introduction with objective, background, and scope and delimitations, main body with problem formulations, derivations/developments and results, conclusions with recommendations for

further work, references, and optional appendices. All figures, tables, and equations shall be numerated. The original contribution of the candidate and material taken from other sources shall be clearly identified. Work from other sources shall be properly acknowledged using quotations and a Harvard citation style (e.g. *natbib* Latex package). The work is expected to be conducted in an honest and ethical manner, without any sort of plagiarism and misconduct. Such practice is taken very seriously by the university and will have consequences. NTNU can use the results freely in research and teaching by proper referencing, unless otherwise agreed upon.

The thesis shall be submitted with a printed and electronic copy to the main supervisor, with the printed copy signed by the candidate, and otherwise according to NTNU procedures. The final revised version of this thesis description must be included. Computer code, pictures, videos, data series, and a PDF version of the report shall be enclosed electronically with all submitted versions.

Start date: 27 August 2018 **Due date:** 21 January 2019

Supervisor: Dong Trong Nguyen
Co-advisor(s): Henrik Schmidt-Didlaukies

Trondheim, _____

_____
Dong Trong Nguyen
Supervisor

Abstract

The idea of using underwater snake robots for subsea operations comes from their ability to perform a wide range of missions, due to the slender and flexible body. Because of these advantages, they are expected to perform more complicated tasks than the more conventional remotely operated vehicles.

The main focus of this thesis is the implementation of an adaptive control algorithm together with an inverse kinematics algorithm to control an underwater snake robot simulator in MATLAB. The need for an adaptive controller is identified as the snake robot might take a range of configurations and interfere with objects subsea. This means that the drag forces, among others, can vary a lot and change rapidly. Such changes are difficult to model and the idea is to have a controller which adapts to these changes. The underwater snake robot should also perform several tasks. By defining the relationship between the snake robot's controllable states and the actual task, the inverse kinematics should be able to calculate the desired motion for the snake robot to achieve the task.

Relevant background on these topics have been presented through a literature review. An adaptive control law has been implemented for the snake robot simulator in this thesis, along with an inverse kinematics algorithm. The theory behind these algorithms is presented, and the process of implementing them is described. To implement these algorithms correctly, theory on kinematics for the snake robot have also been presented.

At last, the system has been simulated for four different cases. Each case defined one or several tasks that should be achieved. The results showed that the adaptive controller worked very well for most cases but had a small issue when the weight of the snake robot was changed to simulate changed dynamics. The inverse kinematics also performed well for the simplest cases but could not always find a feasible solution to achieve several tasks. The conclusion is that the implemented adaptive controller and inverse kinematics could work well for the snake robot, but they need to be further investigated and modified.

Sammendrag

Ideen om bruk av slangeroboter til undervannsoperasjoner handler om dens fleksibilitet og fordelaktige evne til å bevege seg rundt og inne i installasjoner under vann. Slangeroboter er antatt mer egnet for komplekse operasjoner enn mer konvensjonelle undervannsfarkoster.

Hovedfokuset i denne oppgaven er implementasjon av algoritmer for både adaptiv regulering og invers kinematikk for en slangerobot-simulator i MATLAB. Grunnen for at man vil bruke adaptiv regulering for slangeroboten er at den kan ha en veldig varierende dynamikk, da den kan innta flere ulike konfigurasjoner. Det er vanskelig å modellere kreftene på en kompleks robot som har en stadig varierende konfigurasjon. Derfor er det ønskelig å ha en regulator som kan tilpasse seg den stadig varierende dynamikken. Slangeroboter skal kunne utføre ulike oppgaver under vann. Invers kinematikk er viktig for å kunne generere referansene for hvordan slangen faktisk skal bevege seg for å gjennomføre oppgavene den blir satt til å løse.

Et kort litteraturstudie har blitt skrevet om både slangerobotikk, adaptiv regulering og invers kinematikk. Teorien som er nødvendig for implementasjon av algoritmene for invers kinematikk og adaptiv regulering er presentert, etterfulgt av en metodisk gjennomgåelse av hvordan systemet er definert.

Det implementerte systemet har blitt simulert og testet for fire ulike scenarioer. Utfra disse simuleringene kan det konkluderes med at den adaptive reguleringen fungerer svært bra for de aller fleste oppgavene, men har en liten svakhet. Den implementerte invers kinematikken fungerer best for de enkleste scenarioene, men har problemer med å løse flere oppgaver samtidig.

Preface

This master thesis has been written during the fall of 2018 at NTNU (Norwegian University of Science and Technology). The problem description for the thesis was defined together with my supervisor Dong Trong Nguyen, and co-advisor Henrik Schmidt-Didlaukies. The thesis is written on the topic of underwater snake robots and how they can be controlled to perform tasks for subsea operations. Some simulations have been done using MATLAB, and the results are included in the thesis. Several topics will be introduced, and the reader should have some knowledge about engineering to understand the implementation of the different concepts.

Jonas Hopsdal

Acknowledgment

I want to thank my supervisor Dong Trong Nguyen for the help he has provided me when working with this thesis. I would especially like to mention his eagerness to always be available for frequent meetings. My co-advisor Henrik Schmidt-Didlaukies deserves the greatest of thanks, for always being able to clarify the concepts behind the snake robot model, and especially for using countless hours to help me with implementation and debugging.

Contents

List of Tables

List of Figures

Abbreviations/Nomenclature

USM	=	Underwater Swimming Manipulator
USR	=	Underwater Snake Robot
ROV	=	Remotely Operated Vehicle
AUV	=	Autonomous Underwater Vehicle
UVMS	=	Underwater Vehicle-Manipulator System
DOF	=	Degree Of Freedom
IK	=	Inverse Kinematics
ECI	=	Earth-centered inertial reference frame
ECEF	=	Earth-centered Earth-fixed reference frame
NED	=	North East Down reference frame
BODY	=	Reference frame fixed to a moving body
LTI	=	Linear Time Invariant
MRAC	=	Model Reference Adaptive Control
PID	=	Proportional-Integral-Derivative control
$\boldsymbol{\eta}$	-	Position and orientation vector
$\boldsymbol{\eta}_1$	-	Position vector
$\boldsymbol{\eta}_2$	-	Orientation vector
x_b	-	Position of body along inertial x-axis
y_b	-	Position of body along inertial y-axis
z_b	-	Position of body along inertial z-axis
ϕ	-	Roll angle
θ	-	Pitch angle
ψ	-	Yaw angle
$\mathbf{q}$	-	Unit quaternion vector
$\boldsymbol{\Theta}_{ib}$	-	Euler angles of body
$\boldsymbol{\nu}$	-	Body-fixed velocity vector
$\boldsymbol{\nu}_1$	-	Body-fixed translational velocity vector
$\boldsymbol{\nu}_2$	-	Body-fixed angular velocity vector
u	-	Surge velocity
v	-	Sway velocity
w	-	Heave velocity
p	-	Roll rate
q	-	Pitch rate
r	-	Yaw rate
$\mathbf{R}_2^1$	-	Rotation matrix from frame {2} to frame {1}
$\mathbf{T}_\theta$	-	Euler angle transformation matrix
$\mathbf{T}_q$	-	Unit quaternion transformation matrix
$\mathbf{H}_i^{i-1}$	-	Homogeneous tranformation matrix from {i} to {i-1}
$\mathbf{p}_{i/i-1}^{i-1}$	-	Position of {i} relative to {i-1}, expressed in {i-1}
$\mathbf{S}(\cdot)$	-	Skew-symmetric matrix of a vector

$\mathbf{J}$	-	Jacobian matrix
$\boldsymbol{\theta}$	-	Joint position vector
$\boldsymbol{\zeta}$	-	Velocity vector containing base velocities and joint velocities
$\mathbf{M}$	-	Mass matrix
$\mathbf{C}$	-	Coriolis and centripetal matrix
$\mathbf{D}$	-	Drag matrix
$\mathbf{g}$	-	Hydrostatic vector
$\boldsymbol{\tau}$	-	Vector with applied forces to a system
$\boldsymbol{\tau}_b$	-	Forces applied to base
$\boldsymbol{\tau}_\theta$	-	Forces applied to joints
$\boldsymbol{\eta}_d$	-	Desired position and orientation vector
$\boldsymbol{\sigma}_x$	-	Generic task variable
$\mathbf{J}_x$	-	Task jacobian matrix
$(\cdot)^{-1}$	-	Inverse of matrix
$(\cdot)^{+}$	-	Pseudoinverse of matrix
$\mathbf{A}(\mathbf{H})$	-	Adjoint transformation matrix

Chapter 1

Introduction

1.1 Background

Underwater robots have been around for a long time to perform tasks underwater, where ROVs and AUVs are the most commonly used. As the underwater tasks have increased in depth and complexity, the traditional underwater robots have some drawbacks. Especially the common ROV have been deemed unfit for deep operations and close to structures because of the tether. The AUV also might have problems performing tasks close to and inside underwater structures because of its relatively large rigid body.

Snake robots have been identified to work ideal for some subsea operations because of their ability to perform a wide range of operations such as inspections, maintenance and repair. The robots could also be able to navigate close to structures and inside and around pipelines because of their slender body. As the snake robots will work under a range of scenarios and surroundings, the control system must be able to detect changes in surroundings and change the control law or control parameters accordingly. If achieved properly, the robot will be able to operate autonomously. To achieve this, an adaptive controller can be used. The adaptive controller must be stable and robust and at the same time adapt to the identified changes. The underwater snake robot will also need to be able to sort out how it should move in order to perform specific tasks, using inverse kinematics. There has already been done a lot of work on the subjects of inverse kinematics and adaptive control for underwater vehicle-manipulator systems. These systems are similar to snake robots in the sense that they are redundant in terms of the tasks they perform.

1.2 Problem Description

- Perform a background and literature review to provide information and relevant references on:
  - Adaptive Control

  - Snake Robotics
  - Inverse Kinematics

- Write a list with abbreviations and definitions of terms, explaining relevant concepts related to the literature study and project assignment.
- Study a mathematical model of the snake robot
- Study a simulator for the snake robot
- Implement a robust and stable adaptive control algorithm for a snake robot simulator
- Test and validate the proposed adaptive controller by simulations of the snake robot in several scenarios.
- Discuss the approach and results of simulation and conclude whether adaptive control could be useful for controlling a snake robot.

1.3 Outline

This thesis is organized with seven chapters. Chapter 1 is an introduction to the thesis and contains a brief background of the use of underwater snake robots as well as the problem description. Chapter 2 presents a literature review of snake robotics, adaptive control and inverse kinematics. The necessary theory and mathematical background for the implemented adaptive control law and inverse kinematics is presented in Chapter 3. Chapter 4 focuses on how the theory is implemented for the specific system developed as a part of this thesis. This chapter also includes a section on how the implemented controller and inverse kinematics algorithm have been tuned. Chapter 5 presents the four cases which have been set up to test the system, and the results from the simulations are presented. Chapter 6 contains a discussion around the results of the simulations. Chapter 7 is dedicated to the conclusion of this thesis. The MATLAB scripts developed will be listed in the appendix.

Chapter 2

Literature Review

2.1 Snake Robots

The development and research of snake robots is a field that has grown in recent decades. Inspired by the mobility of a real snake and its ability to effectively move through various terrain on land as well as in water, the research aims to mimic the snake motion (Pettersen 2017). Achieving this can be rewarding in terms of meeting unknown environments. However, the snake-like robots are often designed as a series of rigid body parts or "links", which are connected by joints, able to bend in one or several degrees of freedom. This set-up introduces several challenges with regards to modelling the robot and controlling its movements efficiently. Although, if achieved, the robot could be able to reach through tight spaces and create propulsion using its own movements. The first snake robot was developed in 1972 (Liljebäck et al. 2012), and was equipped with wheels at each link, enabling it to move over flat grounds. Since then, much research has been done on the topic.

Most of the early research is related to snake robots on land, not in water. To induce movement on the ground, the research often considers modelling the snake robots with higher tangential friction than normal friction (Shugen Ma et al. 2001). By doing so, the snake robot can create forwards or backwards propulsion by only moving its joints. This property can be achieved by applying passive wheels on the snake robot (Wiriyacharoensunthorn & Laowattana 2002)(Ye et al. 2004). A snake robot without passive wheels has also been developed, (Worst & Linnemann 1996). That specific robot is able to move, as it consists of several sections which all consists of two joints. The joints are composed of aluminum plates which can be bent horizontally or vertically to create movement. By using its own body instead of passive wheels, it is able to creep forward.

(Tanaka & Tanaka 2017) developed a snake robot with passive wheels that can be lifted. The snake utilizes yaw and pitch motion in joints to change the shape of the robot. A method is described to achieve shape control while holding the position and orientation of the snake head, in addition to avoiding self-collision, joint limits and singular configurations.

Studies have also been done for the underwater snake robot (Zuo et al. 2009). For that specific study, a snake robot was developed, where four passive wheels were attached to each link to enable contact with the environment. The robot was simulated to follow serpentine locomotion with different frequencies and amplitudes to find the optimal parameters for giving the most forward speed. The swimming motion of an underwater robot using an eel as inspiration is also discussed by (McIsaac & Ostrowski 2003). The study researches how to define motions that enables swimming forwards, turning and sideways motion.

Some interesting studies have also been done concerning fish robots. One paper utilizes a nonlinear control design for forward propulsion and turning abilities (Morgansen et al. 2001). Another study looks into actuation from fins, with a moving tail section (Morgansen et al. 2007).

The paper, (Tanaka et al. 2015), focuses on collision avoidance for the snake robot body. The system is set up such that the head link velocity is controlled by a joystick while the rest of the snake is controlled such that the head gets its velocity through propulsion from the movement and the robot avoids collision at the same time. This is achieved using sensors that measure the distance from the snake body to some obstacle. This information is fed to a controller which sets the yaw joint velocities by minimizing a cost function.

2.2 Adaptive Control

Technological advances in the field of control theory has allowed more advanced and complex systems to be regulated and controlled. This possibility has created a need for control systems to handle changing and uncertain model dynamics. Adaptive control was first researched in the 1950s, in order to design aircraft autopilots which would function well for different altitudes and speeds (Sastry & Bodson 1989). In other words, adapting to changes. The need for an adaptive controller was identified as a control algorithm with fixed gains could only perform satisfying given a certain environment, but not for changes in the system dynamics. To identify such changes and counteract them was the main objective for developing adaptive controllers at the time. Since then, numerous adaptive control approaches has been proposed to solve the issue for air crafts and other vessels.

A very intuitive approach to regulate systems with changing dynamics is the method of gain scheduling (Sastry & Bodson 1989). This method relies on defining a specifically tuned controller to different possible surroundings. By measuring some auxiliary variables, the controller gains are changed so that they are fit to control the system for the actual dynamics. The drawback of this approach is that the controller relies on good estimations of the surroundings to estimate the dynamics well. The other drawback is that the different gains for the controller must be tuned beforehand. If the system meets an unknown environment, the controller cannot guarantee good performance.

Another approach to adaptive control is called Model Reference Adaptive Control (MRAC) (Sastry & Bodson 1989). This method uses measurements of the actual output of the system and compares it with the desired output. For errors, the control law is updated so that it gives better tracking of the reference.

The adaptive control laws can often be separated as direct or indirect adaptive control laws. Direct adaptive control refers to control laws that uses measurements to directly

change the controller gains or setup, based on the error between the output of the system and the model reference. On the other hand, indirect adaptive control laws uses measurements to first estimate plant parameters, which in turn updates the controller. Such controllers are also called self-tuning regulators. For a LTI system, one approach might be to adjust the system matrices and then calculate controller gains through pole placement (Minamide et al. 1984). For a more complex system, the indirect adaptive control laws might be harder to implement as there is much uncertainty in the dynamics.

For vehicles on sea and below the surface of the sea, the dynamics are non-linear, which means techniques like pole placement are not possible. Several controllers have been proposed to work for such non-linear systems. Variations of the well known PID controller, are usually included for most implemented motion control systems. More advanced control techniques also exists, like linear quadratic optimal control, state feedback linearization, integrator backstepping, adaptive control and sliding mode control (Fossen 2011). Within these techniques there are some different approaches as well, as have already been noted for adaptive control.

There are a few studies which focuses on implementing adaptive control techniques for snake-like robots. An L1 adaptive control algorithm has been simulated and tested for a simulation model of a snake robot (Zhang et al. 2016) to overcome the changing dynamics. Another interesting study is done by (Rollinson & Choset 2013). Here, the snake robot is not controlled by an adaptive controller in that sense, but the behaviour of the snake robot adapts to the environment when performing different maneuvers. This has been demonstrated by making the snake robot climb a pipe with different diameters.

A rather similar area, where a lot of research has been done as well, is for underwater vehicle-manipulator systems. (Antonelli 2013) provides an overview of numerous methods for controlling such systems. Several of them are adaptive control methods. One of the methods proposed (Antonelli et al. 2001), makes use of a system regressor matrix. The control law uses the regressor matrix and the tracking errors to estimate which kind of unknown forces that acts on the system. Hence, this method requires knowledge about the system configuration. Another adaptive control law mentioned, is the non-regressor-based adaptive control law, developed by (Yuh 1996) for UVMSs. As understood by the name, this method does not require much knowledge about the system as it does not require a regressor matrix. The method instead estimates a set of combinations for the bounds of the system matrices. Application of the control law has been discussed in (Lee & Yuh 1999).

2.3 Inverse Kinematics

All systems that are regulated have either a constant reference or a reference trajectory that the system state(s) shall follow. For some systems, with few degrees of freedom or maybe few controllable system states, deciding the references might be easy. For complex systems, there might be some issues regarding the references. If many states are controlled, it might be difficult to visualize how they should move to do some overall task. This is the case for manipulator arms, UVMS and snake robots. These kinds of systems are often highly redundant in the sense that they have more degrees of freedom than necessary to solve simple tasks. A very common task to set for such systems is positioning of the end

effector. In most cases, these types of systems will have a lot of possible configurations that places the end effector at the correct position. However, the end effector position cannot be regulated directly, as its movement is decided by the movement of the joints in the manipulator. The aim for the inverse kinematics is thus to decide the reference trajectories for all the controllable states so that the overall task can be achieved.

Some literature on the topic is from studies of movement for a human-like animations. An iterative method to solve the inverse kinematics was applied in (Erleben & Andrews 2019). The method is numerical and calculates the exact Hessian matrix. In (Moradi & Lee 2005), a closed form method is described for movement of a human-like robotic arm. The movement of the elbow is minimized and joint limits are avoided. The paper concludes that closed form solutions to the inverse kinematics problem should be chosen over iterative methods due to the required time to calculate the references. A linear programming approach is described by (Ho et al. 2005).

(Antonelli 2013), suggests for a UVMS to use a kinematic control approach. With this approach, the velocities of the controllable states are mapped to the velocity of the desired task. The mapping is done by calculating a task specific Jacobian matrix. By inverting the Jacobian matrix, the reference velocities can be calculated for the states. This approach is non-iterative and can thus be calculated quickly. The book also presents some variations on how to invert the Jacobian matrix: the regular pseudoinverse, a weighted pseudoinverse, an augmented Jacobian inverse and finally a Jacobian transpose. It is however noted that these methods does not handle singularities, which might give trouble for calculations. A method based on Lagrange multipliers is described by (Choi 2008). The method describes a singularity-robust solution to the inverse kinematics problem by using the Lagrange multipliers as dampening factors, since they appear close to singular configurations for a redundant manipulator.

A method called task priority redundancy resolution is also presented by (Antonelli 2013). This method allows the inverse kinematics to be solved to achieve several prioritized tasks. The method relies on utilizing the null space of the higher prioritized task Jacobians to achieve the lower prioritized tasks.

Chapter 3

Theory

3.1 Kinematics

As the snake robot consists of many links it is important to be able to express positions and velocities at different places along the snake robot in different reference frames. Knowing the configuration of the snake robot in terms of its links and joints this is possible. Several of the subjects presented in this chapter will be further defined in terms of the actual snake robot model used for this thesis, in chapter 4.

3.1.1 Reference Frames

When working with a system that is moving around in space, defining several reference frames is a convenient way to describe positions and velocities. For a single body, the reference frames used are usually one inertial frame, "attached" to the earth, and a BODY-frame, attached to the some point at the moving object. Then, the translational and angular velocity vectors of the moving object, can be defined in terms of either reference frame. For marine vessels, it is common to use NED-frame (Fossen 2011) and assume it is inertial, which means assuming flat earth. The three axes defining the NED-frame points north (N), east (E) and down (D). The assumption that the earth surface is flat is of course not valid if vessels move far, but is useful for calculating motion in smaller areas where the assumption does not exclude important information. While the moving bodies position is usually defined in the NED-frame, its velocity is usually defined along the axes of the BODY-frame, which moves along with the object.

If however, a vessel move over larger portions of the earth, the Earth-centered Earth-fixed (ECEF) frame or the Earth-centered inertial (ECI) frame is often used. ECEF frame has its origin placed in the center of the earth, and rotates with the natural rotation of the earth meaning that the axes points always points outwards through the earth surface at the same place. The ECI frame is also located in the center of the earth, but it is fixed and does not rotate along with the earths rotation. The constant rotation rate is the only thing that separates the ECEF frame from the ECI frame. A good figure to show the different

reference frames is retrieved from (Fossen 2011), and can be seen in figure 3.1. As seen, the axes of the different frames are denoted with a specific subscript according to the frame they belong to. Note that the subscripts representing each frame in the figure is slightly different from the subscripts that will be used for the snake robot in this thesis.

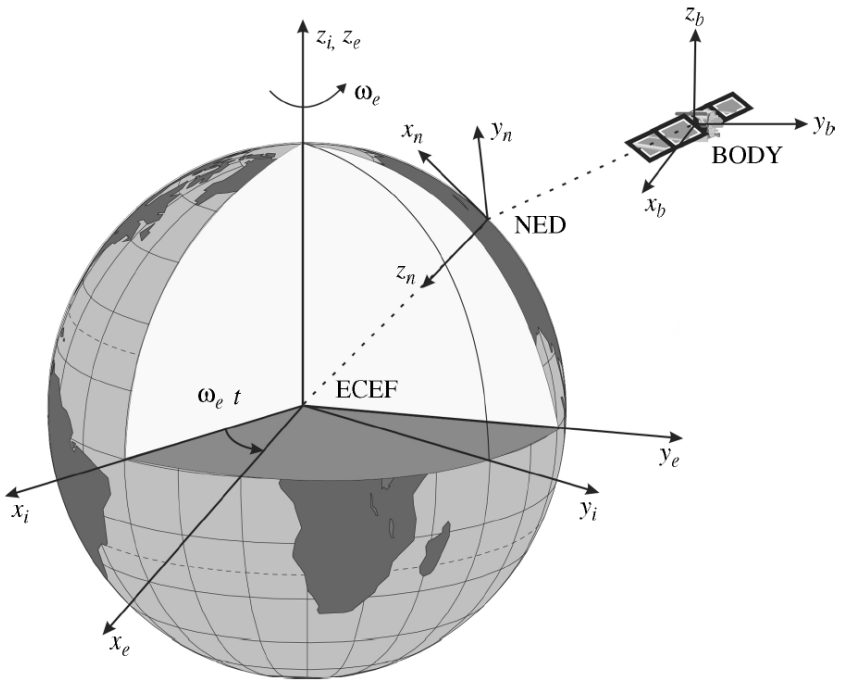

Figure 3.1: An illustration of the different reference frames. The figure is retrieved from (Fossen 2011).

3.1.2 Variables

A marine vessels motion in six degrees of freedom (DOF) is defined by three translational motions and three rotational motions (Fossen 2011). The three translational motions are surge, sway and heave which are defined as the motion along the x-axis, y-axis and z-axis, respectively. The same logic goes for the three rotational motions, which are roll, pitch and yaw. These six degrees of freedom are defined along the axes of the body frame attached to the moving vessel. The velocities for these degrees of freedom are put into the vector $\boldsymbol{\nu}$, which consists of the translational velocity vector, $\boldsymbol{\nu}_1$ (3.1), and the rotational velocity vector, $\boldsymbol{\nu}_2$ (3.2). The velocities in surge, sway and heave are noted u, v and w, respectively. p is the roll rate, q is the pitch rate and r is the yaw rate. The position and orientation of a marine vessel, $\boldsymbol{\eta}$, is defined with respect to one of the inertial frames (3.3). The positions are self explanatory, while ϕ is the roll angle, θ is the pitch angle and ψ is

the yaw angle. This representation of the angles is called the Euler angle representation (Fossen 2011).

$$\boldsymbol{\nu}_1 = [u \quad v \quad w]^\top \tag{3.1}$$

$$\boldsymbol{\nu}_2 = [p \quad q \quad r]^\top \tag{3.2}$$

$$\boldsymbol{\eta} = [x \quad y \quad z \quad \phi \quad \theta \quad \psi]^\top \tag{3.3}$$

The vectors defined above does not explicitly inform about the reference frames the velocities or the positions are defined relative to. To inform about this the velocity vector could be written as $\boldsymbol{\nu} = \boldsymbol{\nu}^b_{b/i}$, which means that it is the velocity of frame b (body frame) relative to i (inertial frame), defined in frame b. The position vector could be written $\boldsymbol{\eta} = \boldsymbol{\eta}^i_{b/i}$, as it is defined in the inertial frame. For the theory presented and applied, this notation will be used, although the shorter version such as in (3.1), (3.2) and (3.3) will be used when the reference frames for the variable have already been defined properly.

3.1.3 Unit Quaternions

The Euler angle representation is the easiest way of understanding the attitude of a moving body. However, it can give trouble with singularities. To overcome this drawback, unit quaternions can be used (Fossen 2011). Unit quaternions uses one real number and three imaginary parameters to represent the three angles in space. It is defined as in equation (3.4). To represent the angles of a body using the unit quaternions, the previously used vector, $\boldsymbol{\eta}_2$, would go from three to four entries and be written $\boldsymbol{\eta}_{2,q}$.

$$\mathbf{q} = \begin{bmatrix} \eta \\ \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix}, \quad \mathbf{q}^T\mathbf{q} = 1 \tag{3.4}$$

3.1.4 Rotation Matrix

When a body is moving around and different reference frames are used, it is important to be able to transform the movements between frames. For example, integrating a moving body's velocity directly would not give it's position in the space where it is moving around. To get the body's position, the velocities would have to be transformed to the inertial frame prior to integration. To do so, the rotation matrix is very useful (Fossen 2011). A rotation matrix can keep track of the angular offsets between any two frames. With the Euler angle representation, the rotation matrix between body and inertial frame is given as in equation (3.5). As the space is three dimensional, the matrix has the dimension 3×3. Note that $c\theta$ and $s\phi$ means $cos(\theta)$ and $sin(\phi)$. The rotation matrix has some interesting properties. It is orthogonal, which means that $\mathbf{R}^\top\mathbf{R} = 1, \mathbf{R}^{-1} = \mathbf{R}^\top$ and $det(\mathbf{R}) = 1$. The translational velocity of the moving body in body frame, $\boldsymbol{\nu}_1$, can then be transformed to the inertial

frame as in equation (3.6). The opposite transformation can be done simply by inverting the rotation matrix (3.7).

$$\begin{aligned}\mathbf{R}_b^i(\boldsymbol{\Theta}_{ib}) &= \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta c\phi \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} & \mathbf{R}_{13} \\ \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{23} \\ \mathbf{R}_{31} & \mathbf{R}_{32} & \mathbf{R}_{33} \end{bmatrix}\end{aligned} \tag{3.5}$$

$$\dot{\boldsymbol{\eta}}_1 = \mathbf{R}_b^i \boldsymbol{\nu}_1 \tag{3.6}$$

$$\boldsymbol{\nu}_1 = \mathbf{R}_b^{i\top} \dot{\boldsymbol{\eta}}_1 \tag{3.7}$$

The rotation matrix can also be found using the unit quaternions (3.8) (Fossen 2011). This rotation matrix is actually important for transformation between the Euler angle representation and unit quaternion representation, because both the Euler angles and unit quaternion can be derived from the rotation matrix. This is possible because the rotation matrices made from either the Euler angles or the unit quaternion vector describes the same rotation. How the Euler angles are retrieved from the rotation matrix is shown in equation (3.9)

$$\begin{aligned}\mathbf{R}_b^i(\mathbf{q}) &= \begin{bmatrix} 1-2(\varepsilon_2^2+\varepsilon_3^2) & 2(\varepsilon_1\varepsilon_2 - \varepsilon_3\eta) & 2(\varepsilon_1\varepsilon_3 + \varepsilon_2\eta) \\ 2(\varepsilon_1\varepsilon_2 + \varepsilon_3\eta) & 1-2(\varepsilon_1^2+\varepsilon_3^2) & 2(\varepsilon_2\varepsilon_3 - \varepsilon_1\eta) \\ 2(\varepsilon_1\varepsilon_3 - \varepsilon_2\eta) & 2(\varepsilon_2\varepsilon_3 + \varepsilon_1\eta) & 1-2(\varepsilon_1^2+\varepsilon_2^2) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} & \mathbf{R}_{13} \\ \mathbf{R}_{21} & \mathbf{R}_{22} & \mathbf{R}_{23} \\ \mathbf{R}_{31} & \mathbf{R}_{32} & \mathbf{R}_{33} \end{bmatrix}\end{aligned} \tag{3.8}$$

$$\phi = tan^{-1}\left(\frac{\boldsymbol{R}_{32}}{\boldsymbol{R}_{33}}\right) \tag{3.9a}$$

$$\theta = -sin^{-1}\left(\boldsymbol{R}_{31}\right) \tag{3.9b}$$

$$\psi = tan^{-1}\left(\frac{\boldsymbol{R}_{21}}{\boldsymbol{R}_{11}}\right) \tag{3.9c}$$

3.1.5 Transforming Angular Velocities

The angular velocities of the moving body must also be transformed to the inertial frame upon integration. This cannot be done through the previously defined rotation matrix, so another transformation matrix is defined (Fossen 2011). The transformation matrix could either be calculated to give the angular velocity of the body in inertial frame as Euler angles (3.10) or unit quaternions (3.12). For Euler angles rates the calculation of

the transformation matrix is presented in equation (3.11) and the transformation matrix for unit quaternion rates is presented in equation (3.13).

$$\dot{\boldsymbol{\Theta}}_{ib} = \mathbf{T}_{\Theta}(\boldsymbol{\Theta}_{ib})\boldsymbol{\nu}_2 \tag{3.10}$$

$$\mathbf{T}_{\Theta}(\boldsymbol{\Theta}_{ib}) = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \tag{3.11}$$

$$\dot{\mathbf{q}} = \mathbf{T}_q(\mathbf{q})\boldsymbol{\nu}_2 \tag{3.12}$$

$$\mathbf{T}_q(\mathbf{q}) = \frac{1}{2}\begin{bmatrix} -\varepsilon_1 & -\varepsilon_2 & -\varepsilon_3 \\ \eta & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & \eta & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & \eta \end{bmatrix} \tag{3.13}$$

3.1.6 Homogeneous Transformation Matrix

Transforming a position in one reference frame to be expressed in another reference frame is a bit different from transforming the translational velocities, which was done only using the rotation matrix between the two frames. The distance between the two frames must also be accounted for. This transformation can be done through a matrix called the homogeneous transformation matrix (3.14) (From et al. 2014). To exemplify its usage, lets say there are two frames, $\{1\}$ and $\{2\}$, and the distance between then is given as the vector $\mathbf{p}^1_{2/1}$. This vector symbolizes the position of frame $\{2\}$ relative to frame $\{1\}$, expressed in $\{1\}$. A point, a, is placed in the space. The vector $\mathbf{p}^2_{a/2}$ points to point a from the origin of frame $\{2\}$. The vector pointing to a from the origin of frame $\{1\}$ can then be found by using the homogeneous transformation matrix (3.15).

$$\mathbf{H}^{i-1}_i = \begin{bmatrix} \mathbf{R}^{i-1}_i & \mathbf{p}^{i-1}_{i,i-1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{3.14}$$

$$\begin{bmatrix} \mathbf{p}^1_{a/1} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}^1_2 & \mathbf{p}^1_{2,1} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \begin{bmatrix} \mathbf{p}^2_{a/2} \\ 1 \end{bmatrix} \tag{3.15}$$

3.1.7 Skew-Symmetric Matrix

When working with vectors, the cross product must often be calculated. These cross products can be calculated using the skew-symmetric matrix, $\mathbf{S}$, which for a vector, $\boldsymbol{\lambda}$, with three entries is defined as in equation (3.16). It can be used to simplify the calculation of the cross product shown in equation (3.17) and due to its skew-symmetry it holds the property of equation (3.18)

$$\mathbf{S}(\boldsymbol{\lambda}) = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix} \tag{3.16}$$

$$\mathbf{a} \times \mathbf{b} = \mathbf{S}(\mathbf{a})\mathbf{b} \tag{3.17}$$

$$\mathbf{S} = -\mathbf{S}^\top \tag{3.18}$$

3.1.8 Jacobian Matrix

The theory behind transforming velocities between reference frames have been shown for simple transformations in sections 3.1.4 and 3.1.5. This can be extended to transforming velocities between a series of reference frames if the position and rotation between them is known. This is done using Jacobian matrices. The Jacobian matrix between a moving body and the inertial frame is just a combination of the rotation matrix and the transformation matrix for angular velocities (3.19) (Fossen 2011).

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu} = \begin{bmatrix} \mathbf{R}_b^i(\boldsymbol{\Theta}_{ib}) & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{T}_\Theta(\boldsymbol{\Theta}_{ib}) \end{bmatrix} \begin{bmatrix} \boldsymbol{\nu}_1 \\ \boldsymbol{\nu}_2 \end{bmatrix} \tag{3.19}$$

3.2 Kinetics

The kinetics of a system defines how external and internal forces are connected to the movement of a system. The movement of the snake robot is, as noted earlier, defined in terms of the system velocity vector, $\boldsymbol{\zeta}$. The vectorial equations of motion for the snake robot is written in equation (3.20). Each of the different terms in the equation represent the different forces acting on the system. The mass and added mass forces are described by the term $\mathbf{M}(\boldsymbol{\theta})\dot{\boldsymbol{\zeta}}$, while the terms $\mathbf{D}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta}$, $\mathbf{C}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta}$ and $\mathbf{g}(\boldsymbol{\theta},\boldsymbol{\eta})$ describes the drag forces, centripetal- and coriolis forces and the hydrostatic forces (Fossen 2011). At last, the term $\boldsymbol{\tau}$ describes the forces applied to the system by a controller.

$$\dot{\boldsymbol{\eta}} = \mathbf{J}(\boldsymbol{\eta})\boldsymbol{\nu} \tag{3.20a}$$

$$\mathbf{M}(\boldsymbol{\theta})\dot{\boldsymbol{\zeta}} + \mathbf{D}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta} + \mathbf{C}(\boldsymbol{\theta},\boldsymbol{\zeta})\boldsymbol{\zeta} + \mathbf{g}(\boldsymbol{\theta},\boldsymbol{\eta}) = \boldsymbol{\tau} \tag{3.20b}$$

3.3 Non-Regressor Based Adaptive Control

The non-regressor based control algorithm which will be presented here was first developed by (Yuh 1996). The method is also referred to as adaptive control with bound estimation (Yuh et al. 1999). The control algorithm aims to estimate combinations of the unknown system parameter matrices, instead of the matrices themselves. The notation used for the control algorithm varies a bit between the sources. In this chapter, the notation is inspired by the notations used in (Yuh et al. 1999) and (Antonelli 2013). The controller will be presented first for an underwater vehicle and secondly for the control of joints.

3.3.1 Vehicle control

First, the kinetic vectorial equation is written in earth-fixed coordinates (3.21). The term $\boldsymbol{u}_d$ is said to be a class of unmodeled effects which are bounded as in equation (3.22). d_0, d_1 and d_2 are positive constants. The system matrices are also assumed to be bounded by the positive constants α_1, β_1, β_2, β_3 and α_2 as shown in equation (3.23). Instead of proving these bounds, the controller will try to estimate a set of combinations between these bounds, as given in equation (3.24). β_4 and β_5 are defined as ε/α, where ε is a positive constant. The two remaining terms d_{-2} and d_{-1} are equal to zero.

$$\mathbf{M}\ddot{\boldsymbol{\eta}} + \mathbf{C}(\boldsymbol{\nu},\boldsymbol{\eta})\dot{\boldsymbol{\eta}} + \mathbf{D}(\boldsymbol{\nu},\boldsymbol{\eta})\dot{\boldsymbol{\eta}} + \mathbf{g}(\boldsymbol{\eta}) + \mathbf{u}_d = \boldsymbol{\tau} \tag{3.21}$$

$$\|\mathbf{u}_d\| \leq d_0 + d_1\|\dot{\mathbf{e}}\| + d_2\|\mathbf{e}\| \tag{3.22}$$

$$\begin{gathered}\left\|\mathbf{M}^{-1}\right\| \leq \alpha, \quad \|\mathbf{M}\| \leq \beta_1, \quad \|\mathbf{C}+\mathbf{D}\| \leq \beta_2, \\ \|\mathbf{g}\| \leq \beta_3, \quad \lambda_{min}(\mathbf{M}^{-1}) > \alpha_2\end{gathered} \tag{3.23}$$

$$\gamma_i = \frac{\alpha(\beta_i + d_{i-3})}{\alpha_2}, \quad i = 1, ..., 5 \tag{3.24}$$

Now, lets define the error of the system as in equation (3.25), where $\boldsymbol{\eta}_d$ is the desired position and orientation of the vehicle. The control law for the vehicle is then presented in equation (3.26), where it can be seen that $\mathbf{\Phi}_1 = \ddot{\boldsymbol{\eta}}_d$, $\mathbf{\Phi}_2 = \dot{\boldsymbol{\eta}}$, $\mathbf{\Phi}_3 = \mathbf{k}$, $\mathbf{\Phi}_4 = \dot{\mathbf{e}}$, $\mathbf{\Phi}_5 = \mathbf{e}$. The vector $\mathbf{k}$ is a psoitive constant vector.

$$\mathbf{e} = \boldsymbol{\eta}_d - \boldsymbol{\eta} \tag{3.25}$$

$$\boldsymbol{\tau}_b = \mathbf{K}_1\ddot{\boldsymbol{\eta}}_d + \mathbf{K}_2\dot{\boldsymbol{\eta}} + \mathbf{K}_3\mathbf{k} + \mathbf{K}_4\dot{\mathbf{e}} + \mathbf{K}_5\mathbf{e} = \sum_{i=1}^{5}\mathbf{K}_i\mathbf{\Phi}_i \tag{3.26}$$

To define the gain matrices $\mathbf{K}_i$ a new error variable is defined, $\mathbf{s}$ (3.27). This error is combination of the velocity error, and the positional error through the positive constant σ. The newly defined error variable will be used when estimating the combinations between the bounds (3.24) (3.28). The constants f_i are the entries of the positive constant adaptation gain vector, $\mathbf{f}$. Now, the estimation of these bounds are used to calculate the gain matrices as shown in equation (3.29).

$$\mathbf{s} = \dot{\mathbf{e}} + \sigma\mathbf{e} \tag{3.27}$$

$$\dot{\hat{\gamma}}_i = f_i\|\mathbf{s}\|\|\mathbf{\Phi}_i\| \tag{3.28}$$

$$\mathbf{K}_i = \frac{\hat{\gamma}_i\mathbf{s}\mathbf{\Phi}_i^\top}{\|\mathbf{s}\|\|\mathbf{\Phi}_i\|} \tag{3.29}$$

The adaptive control algorithm presented above will be proven stable in section 3.3.2, by analysis of a Lyapunov function candidate (Yuh et al. 1999).

3.3.2 Stability Analysis

The Lyapunov function candidate that is evaluated in the stability proof is given in equation (3.30)

$$V = \frac{1}{2}\mathbf{s}^\top \mathbf{s} + \frac{1}{2}\sum_{i=1}^{5} f_i^{-1}\alpha_2(\gamma - \hat{\gamma}_i)^2 \tag{3.30}$$

Remembering that f_i, α_2 and γ_i are all constants, an expression for the Lyapunov function derivative can be found in (3.32), by defining the double derivative of the error (3.31).

$$\begin{aligned}
\ddot{\mathbf{e}} =& \ddot{\boldsymbol{\eta}}_d - \ddot{\boldsymbol{\eta}} \\
=& \ddot{\boldsymbol{\eta}}_d - \mathbf{M}^{-1}(\boldsymbol{\tau}_b - \mathbf{C}\dot{\boldsymbol{\eta}} - \mathbf{D}\dot{\boldsymbol{\eta}} - \mathbf{g} - \mathbf{u}_d) \\
=& \ddot{\boldsymbol{\eta}}_d - \mathbf{M}^{-1}(\mathbf{K}_1\ddot{\boldsymbol{\eta}}_d + \mathbf{K}_2\dot{\boldsymbol{\eta}} + \mathbf{K}_3\mathbf{k} + \mathbf{K}_4\dot{\mathbf{e}} + \mathbf{K}_5\mathbf{e} - \mathbf{C}\dot{\boldsymbol{\eta}} - \mathbf{D}\dot{\boldsymbol{\eta}} - \mathbf{g} - \mathbf{u}_d) \\
=& \mathbf{M}^{-1}(\mathbf{M} - \mathbf{K}_1)\ddot{\boldsymbol{\eta}}_d + \mathbf{M}^{-1}(\mathbf{C} + \mathbf{D} - \mathbf{K}_2)\dot{\boldsymbol{\eta}} + \mathbf{M}^{-1}(\mathbf{g} - \mathbf{K}_3) \\
&+ \mathbf{M}^{-1}\mathbf{u}_d - \mathbf{M}^{-1}\mathbf{K}_4\dot{\mathbf{e}} - \mathbf{M}^{-1}\mathbf{K}_5\mathbf{e} \\
=& \mathbf{M}^{-1}\sum_{i=1}^{5}(\mathbf{P}_i - \mathbf{K}_i)\boldsymbol{\Phi}_i + \mathbf{M}^{-1}\mathbf{u}_d
\end{aligned} \tag{3.31}$$

$$\begin{aligned}
\dot{V} =& \frac{1}{2}\dot{\mathbf{s}}^\top \mathbf{s} + \frac{1}{2}\mathbf{s}^\top \dot{\mathbf{s}} - \sum_{i=1}^{5} f_i^{-1}\alpha_2(\gamma_i - \hat{\gamma}_i)\dot{\hat{\gamma}}_i \\
=& \mathbf{s}^\top(\ddot{\mathbf{e}} + \sigma\dot{\mathbf{e}}) - \sum_{i=1}^{5} f_i^{-1}\alpha_2(\gamma_i - \hat{\gamma}_i)\dot{\hat{\gamma}}_i \\
=& \left[\mathbf{s}^\top(\mathbf{M}^{-1}\sum_{i=1}^{5}\mathbf{P}_i\boldsymbol{\Phi}_i + \mathbf{M}^{-1}\mathbf{u}_d) + \sigma\mathbf{s}^\top\dot{\mathbf{e}} - \sum_{i=1}^{5} f_i^{-1}\alpha_2\gamma_i\dot{\hat{\gamma}}_i\right] \\
&+ \left[-\mathbf{s}^\top\mathbf{M}^{-1}\sum_{i=1}^{5}\mathbf{K}_i\boldsymbol{\Phi}_i + \sum_{i=1}^{5} f_i^{-1}\alpha_2\hat{\gamma}_i\dot{\hat{\gamma}}_i\right]
\end{aligned} \tag{3.32}$$

$$
\begin{aligned}
&\left[\mathbf{s}^\top(\mathbf{M}^{-1}\sum_{i=1}^{5}\mathbf{P}_i\mathbf{\Phi}_i+\mathbf{M}^{-1}\mathbf{u}_d)+\sigma\mathbf{s}^\top\dot{\mathbf{e}}-\sum_{i=1}^{5}f_i^{-1}\alpha_2\gamma_i\dot{\hat{\gamma}}_i\right]\\
=&\mathbf{s}^\top\sum_{i=1}^{3}\mathbf{M}^{-1}\mathbf{P}_i\mathbf{\Phi}_i-\sum_{i=1}^{3}\alpha\beta_i\left\|\mathbf{s}\right\|\left\|\mathbf{\Phi}_i\right\|+\sigma\mathbf{s}^\top\dot{\mathbf{e}}-\varepsilon(\left\|\mathbf{s}\right\|\left\|\dot{\mathbf{e}}\right\|+\left\|\mathbf{s}\right\|\left\|\mathbf{e}\right\|)\\
&+\mathbf{s}^\top\mathbf{M}^{-1}\mathbf{u}_d-\alpha(d_0\left\|\mathbf{s}\right\|+d_1\left\|\mathbf{s}\right\|\left\|\dot{\mathbf{e}}\right\|+d_2\left\|\mathbf{s}\right\|\left\|\mathbf{e}\right\|)\\
\leq&\sum_{i=1}^{3}(\left\|\mathbf{M}^{-1}\right\|\left\|\mathbf{P}_i\right\|-\alpha\beta_i)\left\|\mathbf{s}\right\|\left\|\mathbf{\Phi}_i\right\|+(\sigma-\varepsilon)\left\|\mathbf{s}\right\|\left\|\dot{\mathbf{e}}\right\|-\varepsilon\left\|\mathbf{s}\right\|\left\|\mathbf{e}\right\|\\
&+(\left\|\mathbf{M}^{-1}\right\|-\alpha)(d_0+d_1\left\|\dot{\mathbf{e}}\right\|+d_2\left\|\mathbf{e}\right\|)\left\|\mathbf{s}\right\|\\
\leq&0
\end{aligned}
\tag{3.33}
$$

$$
\begin{aligned}
&\left[-\mathbf{s}^\top\mathbf{M}^{-1}\sum_{i=1}^{5}\mathbf{K}_i\mathbf{\Phi}_i+\sum_{i=1}^{5}f_i^{-1}\alpha_2\hat{\gamma}_i\dot{\hat{\gamma}}_i\right]\\
=&\sum_{i=1}^{5}(-\frac{\mathbf{s}^\top\mathbf{M}^{-1}\mathbf{s}}{\mathbf{s}^\top\mathbf{s}}+\alpha_2)\left\|\mathbf{s}\right\|\left\|\mathbf{\Phi}_i\right\|\hat{\gamma}_i\\
\leq&\sum_{i=1}^{5}(-\lambda_{min}(\mathbf{M}^{-1})+\alpha_2)\left\|\mathbf{s}\right\|\left\|\mathbf{\Phi}_i\right\|\hat{\gamma}_i\\
\leq&0
\end{aligned}
\tag{3.34}
$$

The two brackets found in (3.32) are investigated further in equations (3.33) and (3.34). From the three equations it can be seen that the derivative of the Lyapunov candidate function will be negative for all $\mathbf{s} \neq 0$. Thus, the system is stable and the error $\mathbf{e}$ will converge to zero.

3.3.3 Joint control

The adaptive control algorithm which was presented above can also be implemented for controlling the joints. The joint control law is then given by equation (3.35). Note that the error variables are calculated differently for this controller, as the joints positions and velocities are the source of the errors (3.36) (3.38). The estimations for the combinations of bounds are updated in equation (3.39) and used to define the gain matrices of the control law (3.37).

$$
\boldsymbol{\tau}_\theta=\mathbf{K}_{1,\theta}\ddot{\boldsymbol{\theta}}_d+\mathbf{K}_{2,\theta}\dot{\boldsymbol{\theta}}+\mathbf{K}_{3,\theta}\mathbf{k}_\theta+\mathbf{K}_{4,\theta}\dot{\tilde{\boldsymbol{\theta}}}+\mathbf{K}_{5,\theta}\tilde{\boldsymbol{\theta}}=\sum_{i=1}^{5}\mathbf{K}_{i,\theta}\mathbf{\Phi}_{i,\theta}
\tag{3.35}
$$

$$
\tilde{\boldsymbol{\theta}}=\boldsymbol{\theta}_d-\boldsymbol{\theta}
\tag{3.36}
$$

$$\mathbf{K}_{i,\theta} = \frac{\hat{\gamma}_{i,\theta}\mathbf{s}_\theta \mathbf{\Phi}_{i,\theta}^\top}{\|\mathbf{s}_\theta\| \, \|\mathbf{\Phi}_{i,\theta}\|} \tag{3.37}$$

$$\mathbf{s}_\theta = \dot{\tilde{\boldsymbol{\theta}}} + \sigma_\theta \tilde{\boldsymbol{\theta}} \tag{3.38}$$

$$\dot{\hat{\gamma}}_{i,\theta} = f_{i,\theta} \, \|\mathbf{s}_\theta\| \, \|\mathbf{\Phi}_{i,\theta}\| \tag{3.39}$$

3.4 Inverse Kinematics

An underwater swimming manipulator is kinematically redundant in most cases, as it has more degrees of freedom than needed to perform a maneuver or task. This means that it can move where it is asked to move in different ways. The motivation behind a kinematic control approach is to generate suitable trajectories for the states of the snake robot. The objective of the inverse kinematics is thus to produce reference values for these states such that the overall objective of the snake robot is achieved. These references are passed on to the controller, which aims to make sure the snake robot actually follows these trajectories. The input to the inverse kinematics block can be defined as tasks. Then, it is the objective of the inverse kinematics to make the best use of the systems degrees of freedom to decide how the movements should accomplish the tasks.

A variety of tasks can be defined for a snake robot. Because the idea is for the snake robot to do subsea operations, some useful tasks might include reaching a given configuration of the end effector or the base link, keep the joints in a certain way or to avoid joint limits. These tasks are decided by the operator. Each task is defined by a generic variable $\boldsymbol{\sigma}$ and a task jacobian matrix, $\mathbf{J}_x$. They are defined such that the task Jacobian connects the time derivative of the generic variable to the system velocity vector, $\boldsymbol{\zeta}$, see equation (3.40).

$$\dot{\boldsymbol{\sigma}}_x = \mathbf{J}_x(\boldsymbol{\eta}, \boldsymbol{\theta})\boldsymbol{\zeta} \tag{3.40}$$

When $\dot{\boldsymbol{\sigma}}_x$ is decided by the operator, the reference velocity for the system can be found by inverting the task Jacobian (3.41). However, the task Jacobian might not be a square matrix, hence the pseudoinverse must be calculated (3.42) (Antonelli 2013). This calculation of the pseudoinverse minimizes the square function of the velocity vector (3.43). Another approach is to calculate the weighted pseudoinverse (3.44). This approach is a bit different as it minimizes the function given in equation (3.45).

Some problems might arise when using the pseudoinverse. Singularities can be present if the matrix $\boldsymbol{J}\boldsymbol{J}^\top$ is not full rank.

$$\boldsymbol{\zeta}_r = \mathbf{J}_x^+ \dot{\boldsymbol{\sigma}}_x \tag{3.41}$$

$$\boldsymbol{J}^+ = \boldsymbol{J}^\top \left(\boldsymbol{J}\boldsymbol{J}^\top \right)^{-1} \tag{3.42}$$

$$E = \frac{1}{2}\boldsymbol{\zeta}^\top \boldsymbol{\zeta} \tag{3.43}$$

$$\boldsymbol{J}_W^+ = \mathbf{W}^{-1}\boldsymbol{J}^\top \left(\boldsymbol{J}\mathbf{W}^{-1}\boldsymbol{J}^\top\right)^{-1} \tag{3.44}$$

$$E = \frac{1}{2}\boldsymbol{\zeta}^\top \mathbf{W}\boldsymbol{\zeta} \tag{3.45}$$

The solution when finding the reference velocities in equation (3.41) or (3.44) is only applicable to generate the reference to reach one task. However, for many cases it is possible to do several tasks due to the many degrees of freedom. This can be done by defining a priority of tasks. A simple example would be to define two tasks, a and b, where task a holds top priority over task b. This means that the inverse kinematics should always generate a velocity reference which reaches the first task. The second task is only performed if it does not interfere with the first task. The approach to find such a solution to the inverse kinematics is written in equation (3.46). This solution is the equivalent of letting the second task define reference values in the null space of the first task. That means that the second task might use the degrees of freedom that are not used by the first task. Even more tasks can be put into the priority list. The way to do this is by stacking the top priority task Jacobians when calculating their null space (Antonelli 2013). See equations (3.47), (3.48) and (3.49). Note that all the pseudoinverse jacobians can be calculated using the weighted pseudoinverse as well.

$$\boldsymbol{\zeta}_r = \mathbf{J}_a^+ \dot{\boldsymbol{\sigma}}_a + \left(\mathbf{I} - \mathbf{J}_a^+ \mathbf{J}_a\right) \mathbf{J}_b^+ \dot{\boldsymbol{\sigma}}_b \tag{3.46}$$

$$\boldsymbol{\zeta}_r = \mathbf{J}_a^+ \dot{\boldsymbol{\sigma}}_a + \left(\mathbf{I} - \mathbf{J}_a^+ \mathbf{J}_a\right) \mathbf{J}_b^+ \dot{\boldsymbol{\sigma}}_b + \left(\mathbf{I} - \mathbf{J}_{ab}^+ \mathbf{J}_{ab}\right) \mathbf{J}_c^+ \dot{\boldsymbol{\sigma}}_c \tag{3.47}$$

$$\mathbf{N}_a = \left(\mathbf{I} - \mathbf{J}_a^+ \mathbf{J}_a\right), \quad \mathbf{N}_{ab} = \left(\mathbf{I} - \mathbf{J}_{ab}^+ \mathbf{J}_{ab}\right) \tag{3.48}$$

$$\mathbf{J}_{ab} = \begin{bmatrix} \mathbf{J}_a \\ \mathbf{J}_b \end{bmatrix} \tag{3.49}$$

Chapter 4

Method

4.1 Simulator

The non-regressor based adaptive control algorithm and the algorithm for inverse kinematics are added to the simulator in MATLAB. A qualitative overview of the systems block diagram is presented in figure 4.1. The reference values that are passed from the inverse kinematics block needs to include positional references for the joints and base, velocity references for the joints and base, and finally acceleration references for the joints and base. However, as was shown in section 3.4, the inverse kinematics only computes the velocity references. The way this mismatch is solved is to integrate the velocity references to get the positional references, and differentiate it numerically to get the acceleration references. When doing so, the appropriate mapping between the body frame and inertial frame is done by using the theory of section 3.1.8.

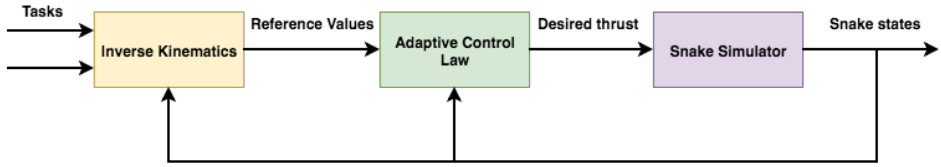

Figure 4.1: Overview of the adaptive controllers and inverse kinematics in closed loop.

4.1.1 Simplifications

There are several simplifications made to the system. In a normal closed loop control system, the forces from the controller could not be applied directly. The normal approach would be to send the desired firces to a thrust allocation block. Then, the thrust allocation would calculate the required force for each of the thrusters mounted on the snake robot. As the snake robot is intended to have thrusters at several links and because the desired forces that the controller block asks for are given at the base link, this could become a

rather complex process. For that reason the snake robot is assumed to have all the required actuation exactly where the controller asks for forces to be applied. As the joints actually would have actuation in the joints, the passing of the desired forces from the joint controller directly on the joints is more realistic than for the base controller.

The other very important assumption that have been made about the system is related to the state feedback to controller and inverse kinematics. In this system, the controllers and inverse kinematics can read which ever state it needs, exactly as the state is. For a more realistic system, this would never be the case. For all real systems, the different states would have to be measured. The measurements are often noisy and would thus not give the exact state of which it tries to measure. This problem would normally be solved by applying a signal processing block, where the measurements would be smoothed out. This process often involves some sort of a filter, which makes sure that the state measurements that are passed on to the other blocks comes without noise and still being as correct as possible. A very commonly used filter for such applications is the Kalman filter (Fossen 2011).

4.1.2 Application in MATLAB

The shell scripts for running a simulation for the snake robot, the script showing a movie of the snake, and all the parts of the simulator calculating the kinetics of the snake robot was provided by the co-advisor, Henrik Schmidt-Didlaukies.

As the whole system is implemented in MATLAB, several scripts have been defined to simplify the coding. The most important scripts are show in figure 4.2, which also shows some of the flow through the code. Some of the script blocks have been colored to inform which scripts belongs to which of the blocks in figure 4.1. At the left side of the figure, are the scripts that takes care of initializing the simulation, solving it at each time step, and finally plotting the results and showing a movie of the snakes behaviour during the simulation. The simulation is solved by looping through each time step until the end of the simulation using the Runge-Kutta Fourth-Order method for integration (4.1) (4.2) (Fossen 2011). On the right side of the figure, all the scripts that uses some of the theory described throughout this these are located. These scripts are ran at each time step. The most important scripts for the implementation in MATLAB can be seen in the appendix, 7.1

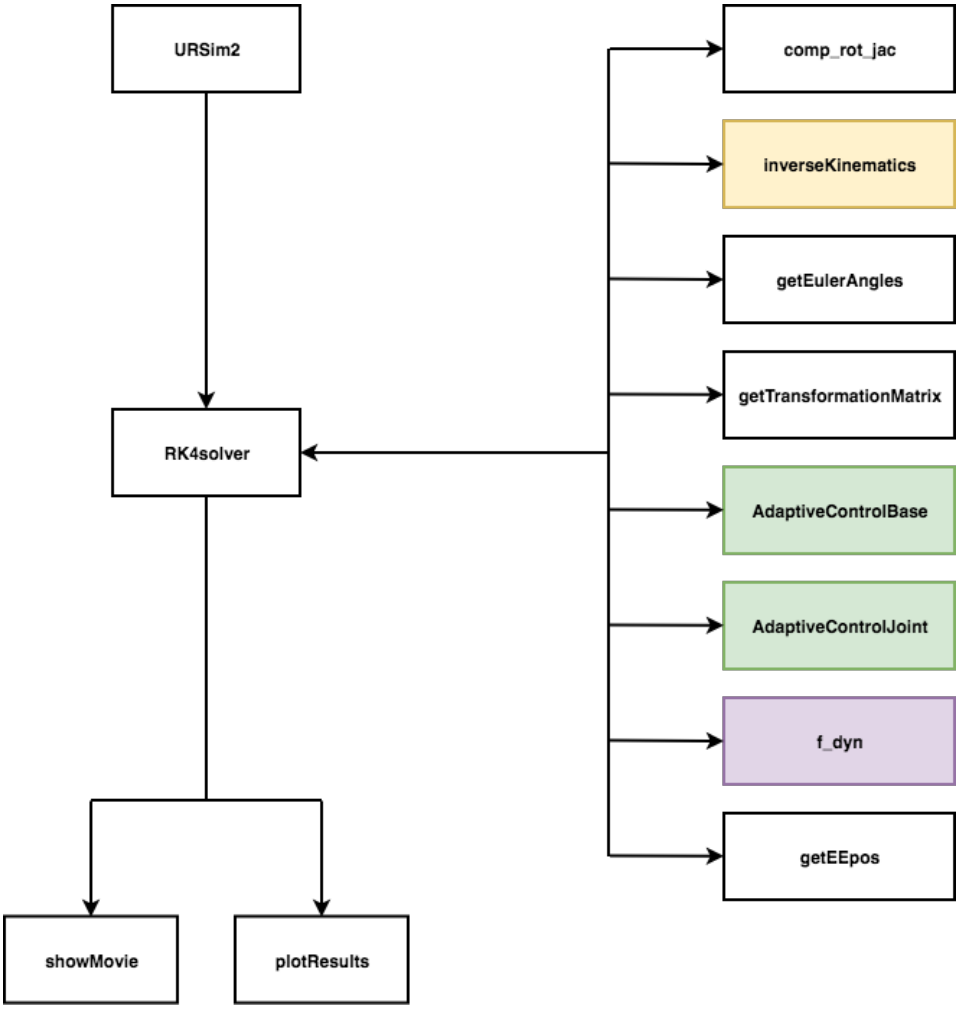

Figure 4.2: Overview of the MATLAB scripts for this system.

$$\begin{aligned}
\mathbf{k}_1 &= h\mathbf{f}(\mathbf{x}(k), \mathbf{u}(k), t_k) \\
\mathbf{k}_2 &= h\mathbf{f}(\mathbf{x}(k) + \mathbf{k}_1/2, \mathbf{u}(k), t_k + h/2) \\
\mathbf{k}_3 &= h\mathbf{f}(\mathbf{x}(k) + \mathbf{k}_2/2, \mathbf{u}(k), t_k + h/2) \\
\mathbf{k}_4 &= h\mathbf{f}(\mathbf{x}(k) + \mathbf{k}_3/2, \mathbf{u}(k), t_k + h)
\end{aligned} \tag{4.1}$$

$$\mathbf{x}(k+1) = \mathbf{x}(k) + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{4.2}$$

4.2 The Snake Robot

The snake robot used for simulations in this thesis has many degrees of freedom as it consists of 6 connected links, so it is important to define the systems reference frames and variables properly. The first link of the snake robot is called the base link, while the last link is called the head link. For subsea operations, a grip arm may often be applied to either end of the snake robot, so the tip of the head link will be referred to as the end effector. See figure 4.3. All of the links are similar, and the joints can rotate with yaw angle relative to each other. The link sizes are listen in table 4.1.

A local reference frame will be defined for each of the links, located at the back end of each link. Thus, the rotation in the joints defines the rotation between the reference frames, and the length of each link defines the distance between them. Figure 4.4 illustrates the placement of the reference frames. The reference frame at the base link will be noted with {b}, while the reference frames for the other links will be numerated 1 to 5, as the snake consists of 6 links. The only reference frame not attached to the snake robot is the inertial frame, which will be written as {i} in subscripts and superscripts.

Table 4.1: Dimension of the links.

Length	Radius
0.75 [m]	0.1 [m]

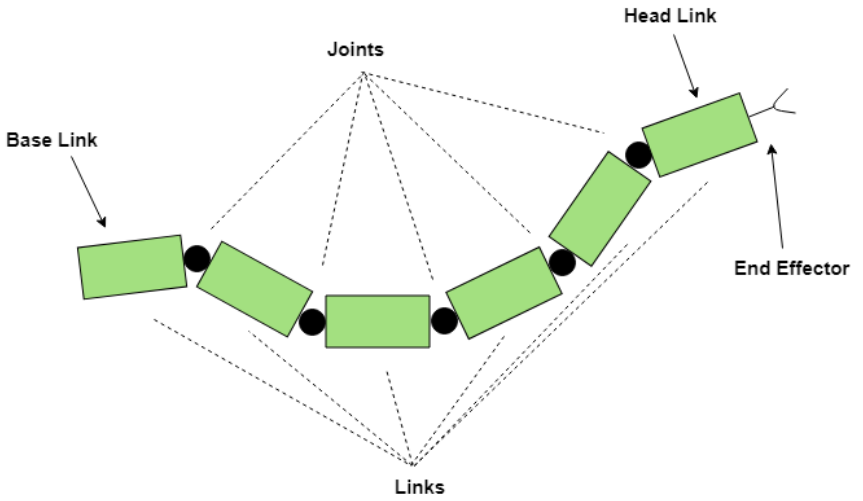

Figure 4.3: Links and joints of the snake robot.

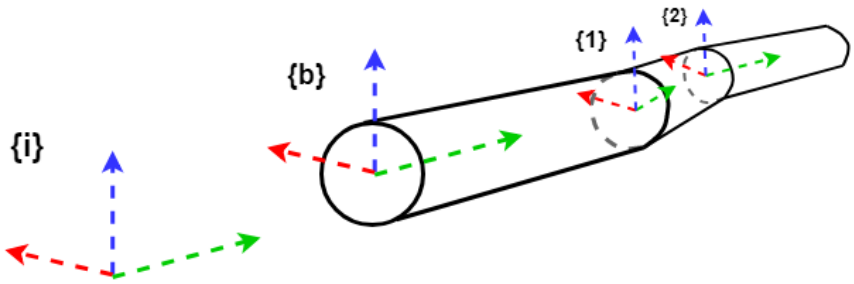

Figure 4.4: Illustration of the snake robots reference frames for the first three links.

As described in section 3.1.2, the body fixed velocity of a moving body is defined as the vector $\boldsymbol{\nu}$. For the snake robot, this vector will note the velocity of the base link in base frame, while the base position is described by the vector $\boldsymbol{\eta}$, which was defined in section 3.1.2. The angular velocities of the joints will be defined as the vector $\dot{\boldsymbol{\theta}}$ (4.3). Thus, the vector $\boldsymbol{\theta}$ represents the angles of the joints. The velocity vector of the base link and the joints velocity vector are combined to make the state velocity vector (4.4).

$$\dot{\boldsymbol{\theta}} = \begin{bmatrix} \dot{\theta}_1 & \dot{\theta}_2 & \dot{\theta}_3 & \dot{\theta}_4 & \dot{\theta}_5 \end{bmatrix}^\top \tag{4.3}$$

$$\boldsymbol{\zeta} = \begin{bmatrix} \boldsymbol{\nu}^\top & \dot{\boldsymbol{\theta}}^\top \end{bmatrix}^\top \tag{4.4}$$

4.3 Forward Kinematics

Calculating positions and orientations at certain places along the snake robot can be done using the homogeneous transformation matrices, which are defined in section 3.1.6. As the distance between each of the frames along the snake robot is the length of the link between then, the positional part, $\mathbf{p}_{j/j-1}^{j-1}$, will only have one non-zero element, which is the links length. Because the frames are attached to the links, the length goes along the x-axis, which gives the link length in the first entry of $\mathbf{p}_{j/j-1}^{j-1}$.

To find the end effectors position and orientation in inertial frame, the position of the end effector relative to the frame attached to the last link must first be defined. In fact, it is similar to the distances between the other frames, as it is just the link length along the x-axis of frame number 5 (4.5). The subscript "ee" means the end effector. Now, the position can be transformed by calculating through all the homogeneous transformation matrices of the system and then adding the position of the base link in inertial frame (4.6). By using the same procedure, the inertial position of other places along the snake can also be expressed from the inertial frame as well.

The orientation of the end effector can be found using a somewhat similar approach, but using the rotation matrix. It is worth noticing that the end effector will have the same orientation as the reference frame on the last link. The rotation matrices between all of the frames can be found as the joint angles and base orientation are known. Then, these rotation matrices are combined into one rotation matrix directly between the inertial frame

and reference frame number 5 (4.7). Now, the orientation of the end effector relative to the inertial frame can be found using equation (3.9). At last, the vector $\boldsymbol{\eta}^i_{ee/i}$ is the combination of the derived position and orientation.

$$\mathbf{p}^5_{ee/5} = [0.75 \quad 0 \quad 0]^\top \tag{4.5}$$

$$\begin{bmatrix} \mathbf{p}^i_{ee/i} \\ 1 \end{bmatrix} = \boldsymbol{\eta} + \mathbf{H}^b_1 \mathbf{H}^1_2 \mathbf{H}^2_3 \mathbf{H}^3_4 \mathbf{H}^4_5 \begin{bmatrix} \mathbf{p}^5_{ee/5} \\ 1 \end{bmatrix} \tag{4.6}$$

$$\mathbf{R}^i_{ee} = \mathbf{R}^i_b \mathbf{R}^b_1 \mathbf{R}^1_2 \mathbf{R}^2_3 \mathbf{R}^3_4 \mathbf{R}^4_5 \tag{4.7}$$

4.4 Jacobian Matrices

For this system, with the velocities of the systems defined in the vector $\boldsymbol{\zeta}$, the aim is to define Jacobian matrices that relates the velocity vector of the system to the velocities of the different locations on the snake robot in inertial frame. To do so, the velocities of the reference frames at the beginning of each link must first be transformed so that they are expressed in the base frame. This will be done by defining a Jacobian matrix for each of the frames fixed to the snake. These Jacobian matrices will relate the velocity of each frame, $1 \leq \mathrm{j} \leq 5$, relative to the base frame, expressed in the base frame, and the system state vector, $\boldsymbol{\zeta}$ (4.8).

The first Jacobian with that functionality is between the base and the base frame, thus the Jacobian is really simple (4.9). For the other frames, it is a bit more complicated, but can be done using the adjoint transformation matrix $\mathbf{A}(\mathbf{H}_j)$ (4.10), where $\mathbf{H}_j$ equals $\mathbf{H}^{j-1}_j$ of section 3.1.6 (From et al. 2014). The mapping can then be written as in equation (4.11), which means that the Jacobian matrices can be defined as in equation (4.12).

$$\dot{\boldsymbol{\eta}}^b_{j/b} = \mathbf{J}_j \boldsymbol{\zeta} \tag{4.8}$$

$$\mathbf{J}_0 = [\mathbf{I}_{6\times 6} \quad \mathbf{0}_{6\times 5}] \tag{4.9}$$

$$\mathbf{A}(\mathbf{H}_j) = \begin{bmatrix} \mathbf{R}^\top_j & -\mathbf{R}^\top_j \mathbf{S}\left(\mathbf{p}^{j-1}_{j/j-1}\right) \\ \mathbf{0}_{3\times 3} & \mathbf{R}^\top_j \end{bmatrix} \tag{4.10}$$

$$\dot{\boldsymbol{\eta}}^b_{j/b} = \mathbf{A}(\mathbf{H}_{j-1}) \mathbf{J}_{j-1} \boldsymbol{\zeta} + [\mathbf{0}_{1\times 5} \quad 1]^\top \dot{\theta}_{j-1} \tag{4.11}$$

$$\mathbf{J}_j = \mathbf{A}(\mathbf{H}_{j-1}) \mathbf{J}_{j-1} + \begin{bmatrix} & \mathbf{0}_{5\times 11} & \\ \mathbf{0}_{1\times(6+j-1)} & 1 & \mathbf{0}_{1\times(11-6-j)} \end{bmatrix} \tag{4.12}$$

The upper three rows of the Jacobian matrix maps the translational velocities, while the lower three maps the rotational velocities. By using the mappings from sections 3.1.4 and 3.1.5, the velocities can be transformed out to inertial frame as well. But first, the Jacobian is split into the two parts, $\mathbf{J}_{j,translational}$ and $\mathbf{J}_{j,rotational}$ (4.13). The velocities

of the links reference frames in the inertial frame can then be described as in equation (4.14)

$$\mathbf{J}_j = \begin{bmatrix} \mathbf{J}_{j,translational} \\ \mathbf{J}_{j,rotational} \end{bmatrix} \tag{4.13}$$

$$\dot{\boldsymbol{\eta}}^i_{j/i} = \mathbf{J}^i_j \boldsymbol{\zeta} = \begin{bmatrix} \mathbf{R}^i_b \mathbf{J}_{j,translational} \\ \mathbf{T}_\Theta \mathbf{J}_{j,rotational} \end{bmatrix} \boldsymbol{\zeta} \tag{4.14}$$

4.5 Modifications to Adaptive Control Laws

The adaptive control algorithms presented in section 3.3 are defined from a very theoretical point of view, so that the stability can be proved. However, there are some drawbacks when implementing the controllers. The first obvious problem is that the adaptive update of the estimate of the system matrices set of combinations, never decreases. The equation is repeated here for simplicity (4.15). As seen, the norms of the vectors $\mathbf{s}$ and $\boldsymbol{\Phi}_i$ will always be positive. The adaptation gains, f_i, are also all positive. Thus, the estimations $\hat{\gamma}_i$, will only grow larger as the time goes. This is problematic as the gain matrices of the control law might get very large. The solution to this problem is to let the adaptive update decrease with some rate, ε_i, if the norms are smaller than some threshold, μ_i (Lee & Yuh 1999). See equation (4.16).

$$\dot{\hat{\gamma}}_i = f_i \left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\| \tag{4.15}$$

$$\begin{aligned} \dot{\hat{\gamma}}_i &= f_i \left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\|, \quad \left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\| \geq \mu_i \\ \dot{\hat{\gamma}}_i &= -\varepsilon_i \hat{\gamma}_i + f_i \left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\|, \quad \left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\| < \mu_i \end{aligned} \tag{4.16}$$

The second problematic aspect of implementing the adaptive control algorithm is that the gain matrices might get very large as the error gets low. The calculation of the gain matrices is repeated in equation (4.17). To solve this, another set of thresholds are introduced, δ_i, which are used if the product of the norms of $\mathbf{s}$ and $\boldsymbol{\Phi}_i$ is small enough. The implementation of this modification is shown in equation (4.18) (Yuh et al. 1999).

$$\mathbf{K}_i = \frac{\hat{\gamma}_i \mathbf{s} \boldsymbol{\Phi}_i^\top}{\left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\|} \tag{4.17}$$

$$\begin{aligned} \mathbf{K}_i &= \frac{\hat{\gamma}_i \mathbf{s} \boldsymbol{\Phi}_i^\top}{\left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\|}, \quad \left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\| \geq \delta_i \\ \mathbf{K}_i &= \frac{\hat{\gamma}_i \mathbf{s} \boldsymbol{\Phi}_i^\top}{\delta_i}, \quad \left\| \mathbf{s} \right\| \left\| \boldsymbol{\Phi}_i \right\| < \delta_i \end{aligned} \tag{4.18}$$

The last problem that was experienced with the implementation of the controllers was concerning the joint controller. Severe oscillations were experienced in increasing order from joint 1 to 5. The solution to this problem was to lower the magnitude of the error

variable, $\mathbf{s}$. This was done in a way so that the second entry of $\mathbf{s}$ was lowered more than the first entry. The third entry was lowered more than the second, and so on. To implement this, a vector $\boldsymbol{\epsilon}$ was introduced. $\mathbf{s}$ was then multiplied element-wise with $\boldsymbol{\epsilon}$. A similar solution was applied to the base controller as well.

4.6 Tasks for Inverse Kinematics

For this system, four tasks have been defined. Each task is defined by its desired position for some state of the snake robot along with the corresponding Jacobian, connecting the task to the velocity vector. As each task has been defined as positions and not velocities, changes have been made when calculating reference velocity. Instead of using the velocity of the generic variable, $\dot{\boldsymbol{\sigma}}_x$, a vector $\mathbf{w}_x$ is used, which combines both $\dot{\boldsymbol{\sigma}}_x$ and $\boldsymbol{\sigma}_x$ (4.19) (Antonelli 2013). From the equation it is seen that the positional error of the task is added to the term. This way, the inverse kinematics will work when the system runs closed loop, giving reference velocities that ensures that the error goes to zero.

$$\mathbf{w}_x = \dot{\boldsymbol{\sigma}}_x + \mathbf{K}_{task}\tilde{\boldsymbol{\sigma}}_x \tag{4.19}$$

4.6.1 End Effector Configuration

The end effector is important to control when doing subsea operations. This task defines configuration of the end effector in the inertial frame, meaning its position and orientation. Thus, the task variable is given as in equation (4.20). The subscript d means that it is the desired configuration and should be chosen by the operator.

From equation (4.14), the relation between the velocity of the different reference frames on the snake robot given in inertial was related to the velocity vector, $\boldsymbol{\zeta}$. By choosing the last Jacobian from that equation, the Jacobian for this task is then given as in equation (4.21).

$$\boldsymbol{\sigma}_{ee} = \boldsymbol{\eta}^i_{ee/i,d} \tag{4.20}$$

$$\mathbf{J}_{ee} = \mathbf{J}^i_5 \tag{4.21}$$

4.6.2 Base Link Configuration

This task has many similarities with the end effector task. By the same reasoning, the task variable and Jacobian matrix are shown in equations (4.22) and (4.23).

$$\boldsymbol{\sigma}_{base} = \boldsymbol{\eta}^i_{b/i,d} \tag{4.22}$$

$$\mathbf{J}_{base} = \mathbf{J}^i_b \tag{4.23}$$

4.6.3 Nominal Configuration

This task is a bit different from the first two tasks, as it does not set the position for any of the snake robots parts relative to the inertial frame. The nominal configuration task aims only to keep the joints at some predetermined angle. Thus, the task variable is given in equation (4.24) and the task Jacobian is given in equation (4.25). As seen, the task Jacobian becomes very simple for this case as it only sets parameters that are already in the velocity vector, $\boldsymbol{\zeta}$. It is worth noticing that this task Jacobian have different dimensions compared to the previous task Jacobians.

$$\boldsymbol{\sigma}_{nominal} = \boldsymbol{\theta}_d \tag{4.24}$$

$$\mathbf{J}_{nominal} = \begin{bmatrix} \mathbf{0}_{5\times 6} & \mathbf{I}_{5\times 5} \end{bmatrix} \tag{4.25}$$

4.6.4 Joint Limit Task

The aim of this task is quite different from all the other tasks. Where the other tasks are active the whole time, this task only activates if the joint angles goes outside the accepted range of angles, which is set by this task. The way this is implemented is by first setting a limit for the joint angles (both positive and negative) (4.26). If all the joints are within the accepted range, the task variable is a vector with only zeros in all entries. The task Jacobian is also just a zero matrix. However, if joint number j exceeds the limit, the j'th entry of the task variable vector is set to the joint limit that was exceeded. This means that if the joint angle is currently $\pi/2$ and the joint limits are $\pm\pi/3$, the j'th entry of the task variable would be set to $\pi/3$. At the same time, the entry at column number 6+j, and row number j, is set to 1. This way the task activates and seeks to drive the angle back to the limit. Once the angle is back within the accepted range, the specific entries of the task variable and task Jacobian are set to zero again. In many ways, when the task is activated, it has the same behaviour of the nominal configuration, except that it has two possible desired angles. If all joints have exceeded either limit, the task variable and task Jacobian would look like in equations (4.27) and (4.28).

$$\theta_{limit} = \pm\frac{\pi}{3} \tag{4.26}$$

$$\boldsymbol{\sigma}_{jointlimit} = \begin{bmatrix} \theta_{limit} \\ \theta_{limit} \\ \theta_{limit} \\ \theta_{limit} \\ \theta_{limit} \end{bmatrix} \tag{4.27}$$

$$\mathbf{J}_{jointlimit} = \begin{bmatrix} \mathbf{0}_{5\times 6} & \mathbf{I}_{5\times 5} \end{bmatrix} \tag{4.28}$$

4.7 Tuning of system parameters

The full system, including tasks, inverse kinematics and controllers, contains numerous constants which are to be set by the designer of the system. To tune the parameters of the control laws, a simple reference trajectory is generated. The reason the controllers are tuned without the inverse kinematics block in the loop is to avoid the chance of bad interaction between the inverse kinematics and the control laws when none of the parameters in the system have been tuned properly. It should be easier to tune the two blocks independently at first, then see how well the full system functions when both blocks are included in the loop.

4.7.1 Tuning of adaptive controllers

The reference for the control system is split in two parts. Notice that the snake robot here consists of 6 links connected by 5 joints, which means the reference will have dimension 11×1. At the start of the simulation the reference velocity for the base link is set as in equation (4.29), meaning the reference has an acceleration of 0.1 along the x-axis and 0.05 along the y-axis of the base frame. All the other motions should be zero. Five seconds into the simulation the reference changes to equation (4.30). From a practical point of view, this means the snake robot should move forwards and a bit sideways to start with, then keep the base link still while the joints move back and forth. Notice that t is the time.

$$\boldsymbol{\zeta}_{ref} = \begin{bmatrix} 0.1t & 0.05t & \mathbf{O}_{1\times 9} \end{bmatrix}^\top \tag{4.29}$$

$$\boldsymbol{\zeta}_{ref} = \begin{bmatrix} \mathbf{O}_{6\times 1} \\ 0.05 \cdot \sin(0.4t) \\ 0.05 \cdot \sin(0.4t + 0.2) \\ 0.05 \cdot \sin(0.4t + 0.4) \\ 0.05 \cdot \sin(0.4t + 0.6) \\ 0.05 \cdot \sin(0.4t + 0.8) \end{bmatrix} \tag{4.30}$$

The parameters for the base and joint controllers that should be tuned, are $\mathbf{f}$, $\mathbf{k}$, $\boldsymbol{\delta}$, $\boldsymbol{\sigma}$, $\boldsymbol{\mu}$ and $\boldsymbol{\epsilon}$. Scaling of the variable $\mathbf{s}$, by $\boldsymbol{\epsilon}$, is also performed during the tuning process. As a starting point for tuning the parameters, (Lee & Yuh 1999), gave some insight to how the relative sizes of the parameters could be chosen.

During the process of deciding the values for all the parameters in the control laws, the controllers showed an interesting behaviour. Some of the parameters could easily give destabilizing oscillations to the system if they were set too high or too low. This was the case for δ, $\boldsymbol{\epsilon}$ and $\mathbf{k}$. If δ was set to low values, the controllers would start oscillating. This happens because the controller gains, $\mathbf{K}_i$, becomes very large when the tracking error becomes small. On the other hand, if it was set too high the controller would not be able to achieve good tracking of the reference. The same was experienced for the choice of $\mathbf{k}$. The tracking was bad for low values, while large values resulted in oscillations. The system was also experiencing severe oscillations if $\mathbf{s}$ was not scaled down through the vector $\boldsymbol{\epsilon}$.

The final values of the parameters were chosen to be as in tables 4.2 and 4.3. Figures 4.5, 4.6 and 4.7 shows the tracking the controllers were able to achieve with this choice

of parameters. As can be seen in the figures, the base controller is able to follow the reference trajectory for the position almost perfectly. Considering the orientation of the base link, some very small undesired motions can be noticed. These small motions might be a result of the coupled drag motions the snake robot is experiencing when it starts to move. Even though the base frame is aligned with the inertial frame initially, these figures allows for one important observation. As the base link experiences small motion in pitch angle, the reference for position in heave starts to decrease. This is only natural as the base link will keep tracking the reference velocity which was given in its BODY-frame. Thus, when transforming the desired BODY-fixed velocity to the inertial frame, it gives a small velocity downwards in the inertial frame. The same argument goes for the connection between a small roll angle and the desired speed in sway.

The tracking for the adaptive controller of the snake robot joints is also satisfactory in this case. The tracking is a bit less accurate than for the base link position. This might be explained from the fact that all the joints start moving and follow their trajectories at the same time. Their movements induce forces on each other. This also results in offsets for the base link, where the controller only aims to hold the base link still after the first five seconds of the simulation. This can be seen for the yaw angle of the base link, figure 4.6. It struggles with some small motions, due to all the movements in the joints.

Table 4.2: Tuned parameters for the adaptive controller of the base link.

Parameter	Value
σ	10
$\mathbf{f}$	$[3 \quad 3 \quad 3 \quad 1 \quad 1]^\top$
$\mathbf{k}$	$[1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^\top$
$\boldsymbol{\epsilon}$	$[1 \quad 1 \quad 1 \quad 0.2 \quad 0.2 \quad 0.2]^\top$
$\boldsymbol{\mu}$	$[0.5 \quad 0.5 \quad 2 \quad 0.1 \quad 0.1]^\top$
$\boldsymbol{\varepsilon}$	$[0.05 \quad 0.05 \quad 0.05 \quad 0.02 \quad 0.02]^\top$
δ	10

Table 4.3: Tuned parameters for the adaptive controller of the joint angles.

Parameter	Value
σ	10
$\mathbf{f}$	$[10 \quad 10 \quad 10 \quad 3 \quad 3]^\top$
$\mathbf{k}$	$[3 \quad 3 \quad 3 \quad 3 \quad 3]^\top$
$\boldsymbol{\epsilon}$	$[0.5 \quad 0.25 \quad 0.125 \quad 0.0625 \quad 0.03125]^\top$
$\boldsymbol{\mu}$	$[0.4 \quad 0.4 \quad 2 \quad 0.2 \quad 0.2]^\top$
$\boldsymbol{\varepsilon}$	$[0.05 \quad 0.05 \quad 0.05 \quad 0.02 \quad 0.02]^\top$
δ	

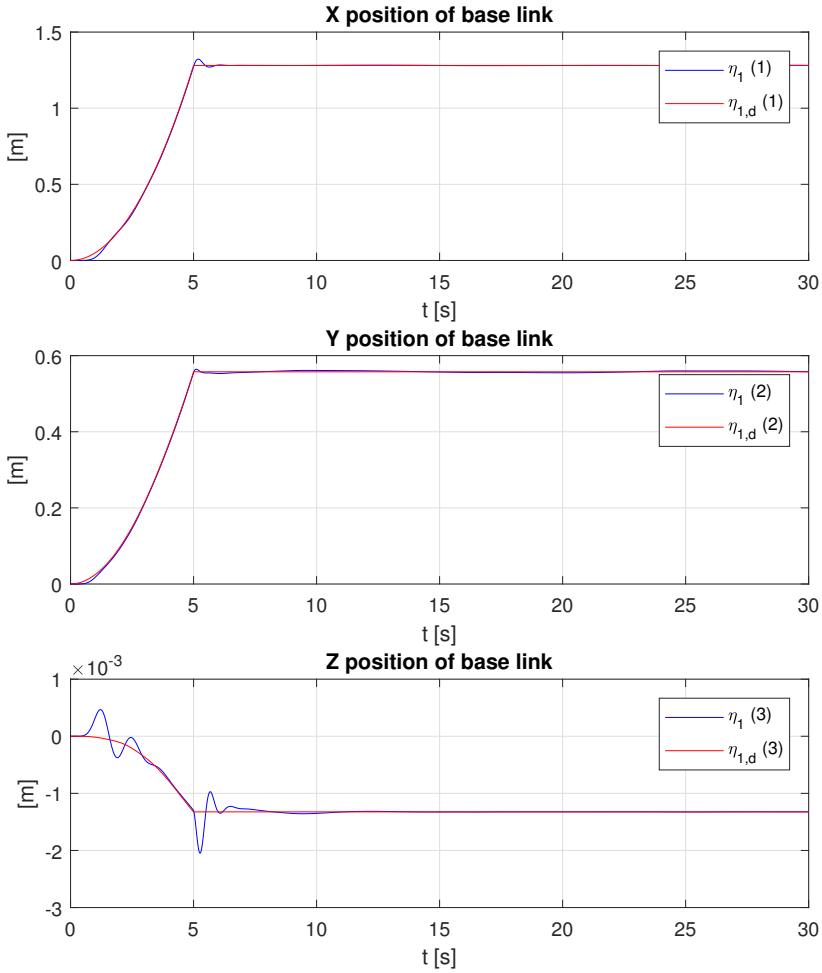

Figure 4.5: Base link position in inertial frame for final simulation of parameter tuning.

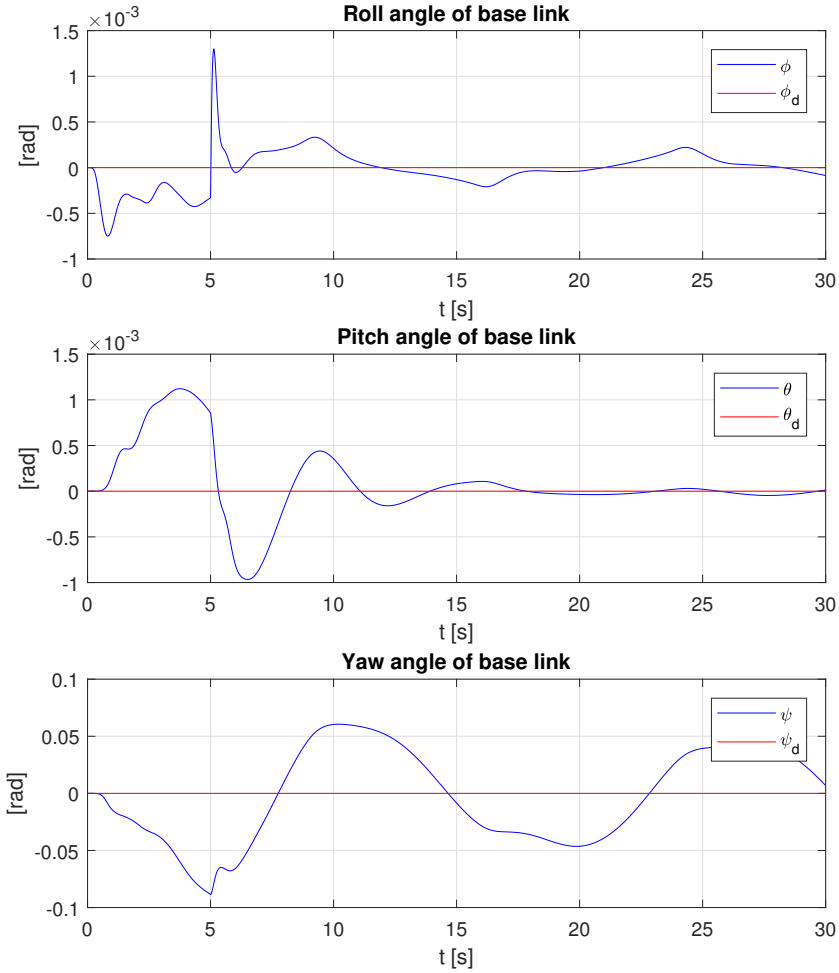

Figure 4.6: Base link orientation for final simulation of parameter tuning.

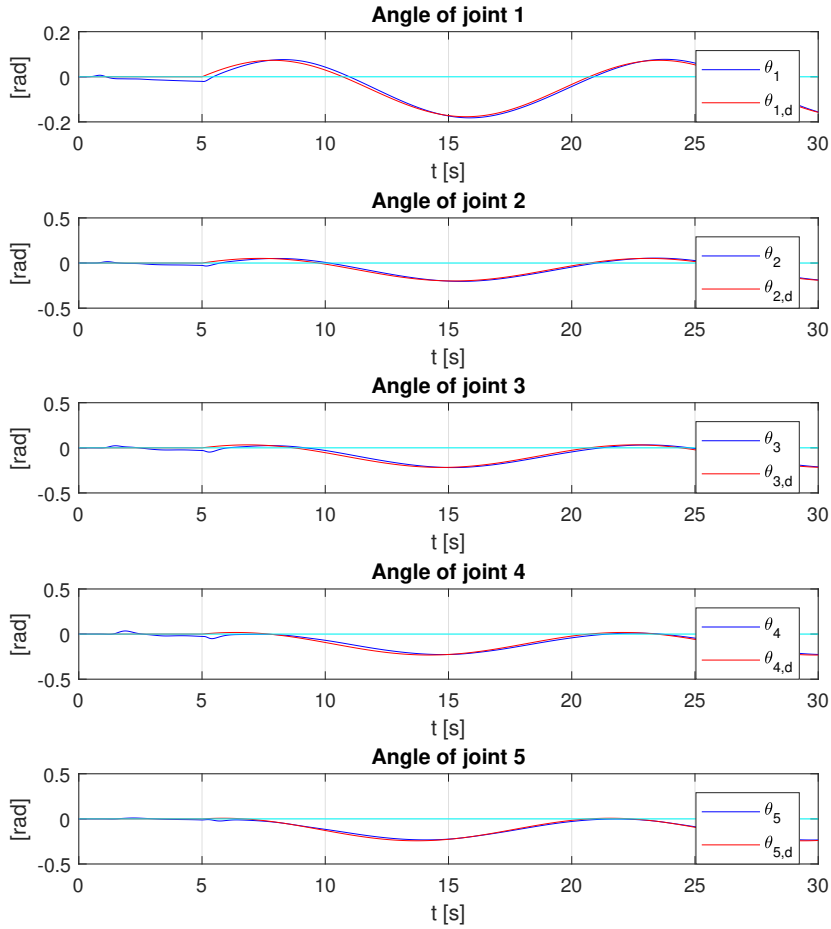

Figure 4.7: Joint angles for final simulation of parameter tuning.

4.7.2 Choice of parameters for inverse kinematics

For the inverse kinematics, the only parameters that can be tuned are the $\mathbf{K}_{task}$ matrices related to each task. As this parameter decides something about the aggressiveness of how fast the reference will reach the tasks, it is very important that it is not chosen too large. It is important that the controllers can track the reference faster than the reference changes. If that is not the case and the reference changes faster than the controller is able to follow, the system might quickly destabilize as the controller output would grow larger and larger as it never reaches the reference. This was tested and the chosen gains, $\mathbf{K}_{task}$, for the

different tasks are shown in table 4.4.

The other possibility to change the dynamics of the inverse kinematics is related to the choice of tasks and their priority. No general choice is made here as the tasks will be prioritized and used differently for achieving the objectives of the different simulated cases in the next chapter. The last option to change the inverse kinematics, concerns the pseudo inverses of the task Jacobians. The system allows for either weighted pseudo inverse 3.42 or just the regular pseudo inverse 3.44.

Table 4.4: Matrix gains for the different tasks.

Task	Associated matrix gain, $\mathbf{K}_{task}$
End effector configuration	diag([0.2 , 0.2 , 0.2 , 0.2 , 0.2 , 0.2])
Base link configuration	diag([0.2 , 0.2 , 0.2 , 0.2 , 0.2 , 0.2])
Nominal configuration	diag([0.2 , 0.2 , 0.2 , 0.2 , 0.2 , 0.2])
Joint limit	diag([0.2 , 0.2 , 0.2 , 0.2 , 0.2 , 0.2])

Chapter 5

Results

This chapter contains results for four different cases that have been simulated to show how the implemented system works. All the simulations were initialized with the base link located at the origin of the inertial frame and with zero angle for all joints, see figure 5.1. Each simulation lasted for 30 seconds. The parameters for the adaptive controllers used in the simulations are the tuned parameters from tables 4.2 and 4.3, while the parameters for the tasks are chosen as in table 4.4. For the first case, the regular pseudo inverse is used in the inverse kinematics block. The three last cases use the weighted pseudo inverse (3.44) with the mass matrix as weighting.

Tasks and any special differences made to the system for the different simulations is noted in the section for each case. Relevant plots are included as well as a picture of the snake robot's configuration at the end of each simulation. The results will be discussed in chapter 6.

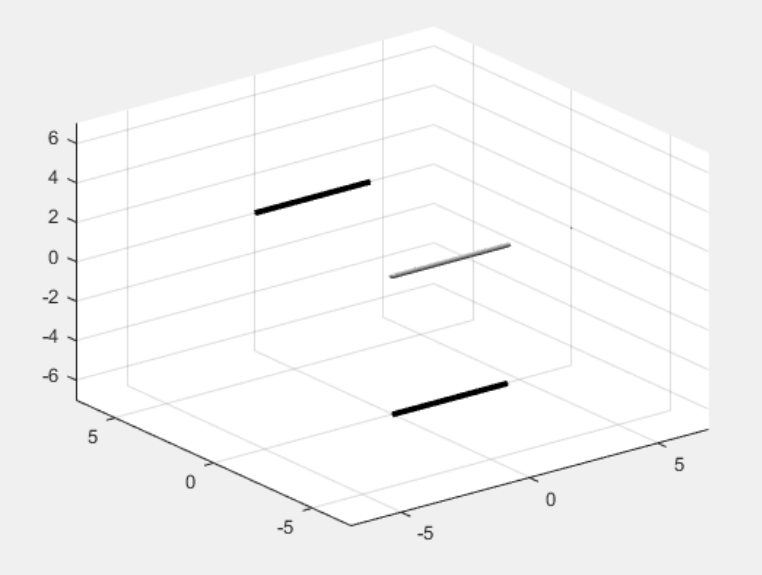

Figure 5.1: Initial position of the snake robot for all simulations.

5.1 Case A

The first simulation done to show how the full system works, aims to achieve a simple base link configuration task. The regular pseudo inverse is used to solve the inverse kinematics and calculate the reference. The system aims to place the base link such that its position and orientation gets to the point set by the task, see equation (5.1). The achieved configuration of the snake robot from the end of the simulation is visualized in figure 5.2, while figures 5.3, 5.4 and 5.5 shows plots for the base links position and orientation as well as the joint angles throughout the simulation.

$$\boldsymbol{\eta}_{base,task} = [0 \quad 2 \quad 0 \quad 0 \quad 0 \quad -\pi/2]^\top \tag{5.1}$$

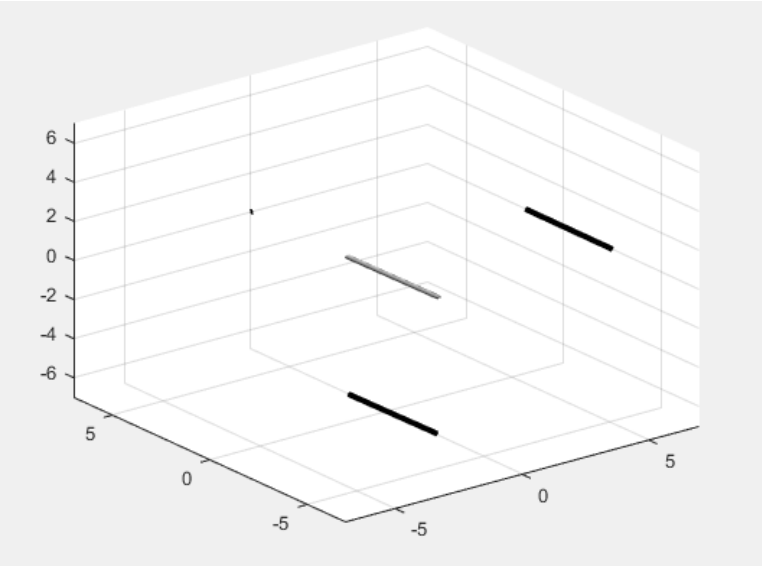

Figure 5.2: Case A: Snake position at end of simulation.

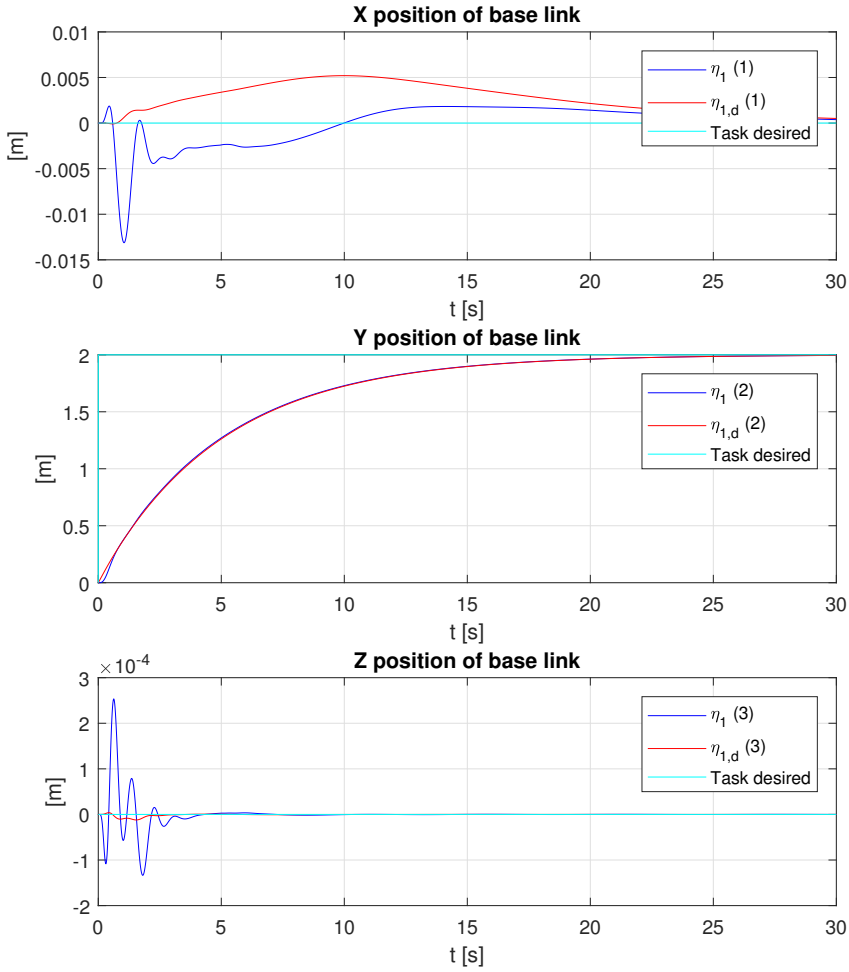

Figure 5.3: Case A: Base link position in inertial frame.

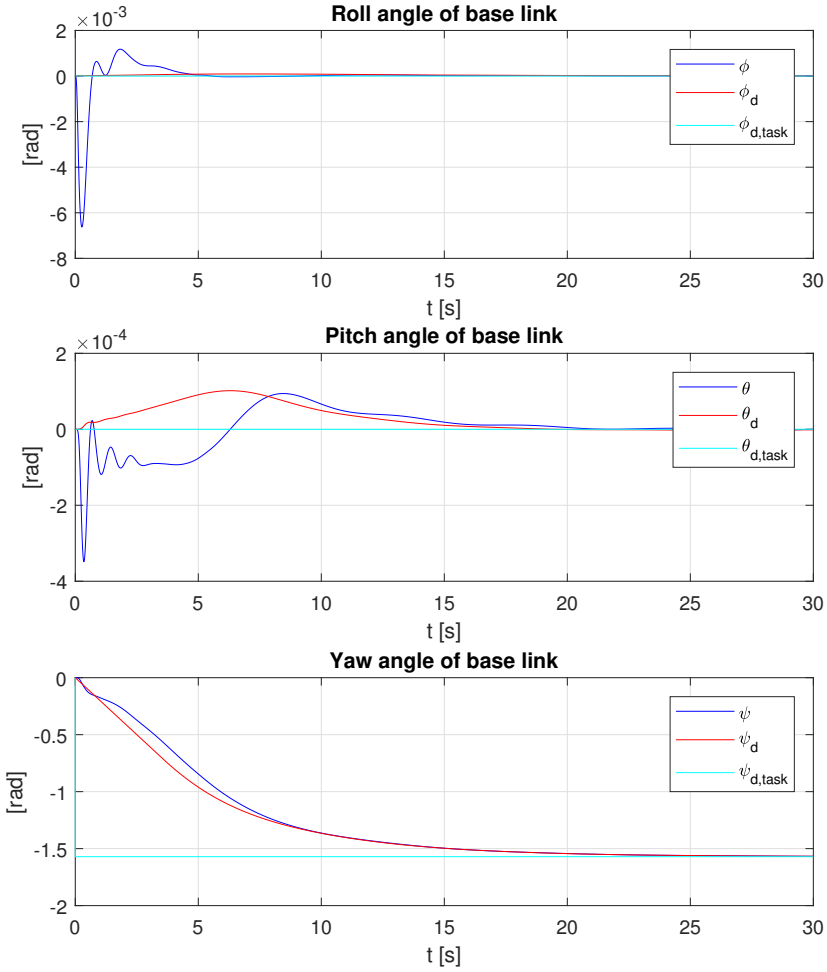

Figure 5.4: Case A: Base link orientation with reference to inertial frame.

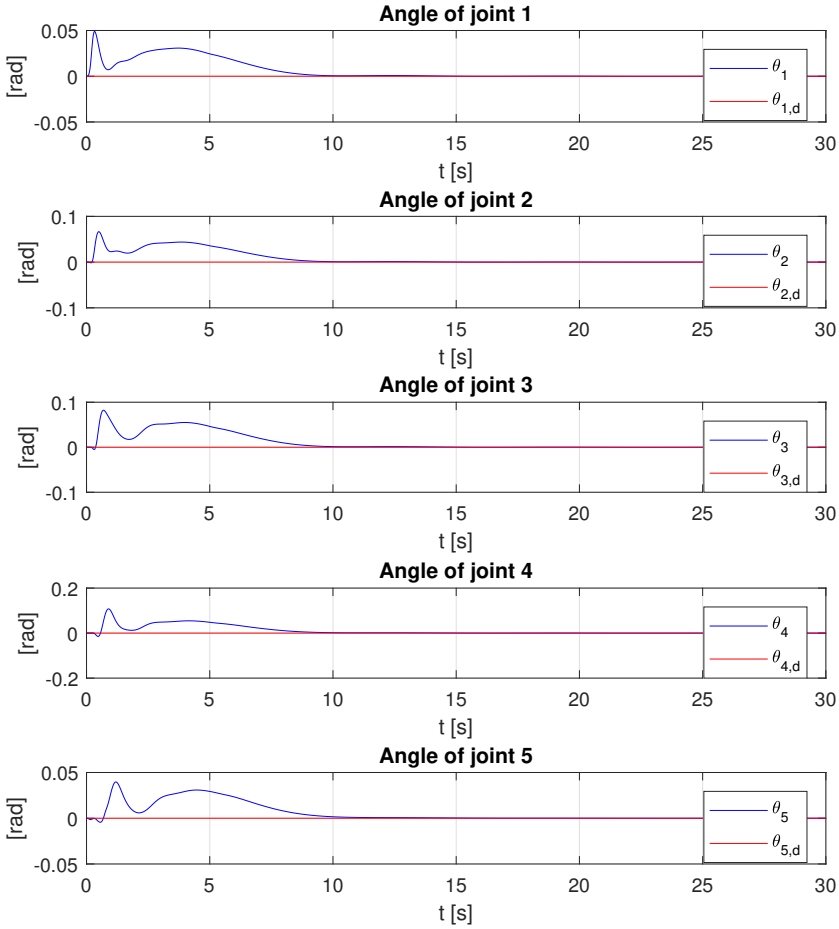

Figure 5.5: Case A: Joint angles.

5.2 Case B

Similar to case A of section 5.1, this case also has the task of achieving a given position and orientation for the base link. The only difference is that the weighted pseudoinverse is used within the inverse kinematics 3.44. The final position of the snake robot after simulation can be seen in figure 5.6. Figure 5.7 shows the base links position, figure 5.8 shows the base link orientation and figure 5.9 shows the joint angles throughout the simulation.

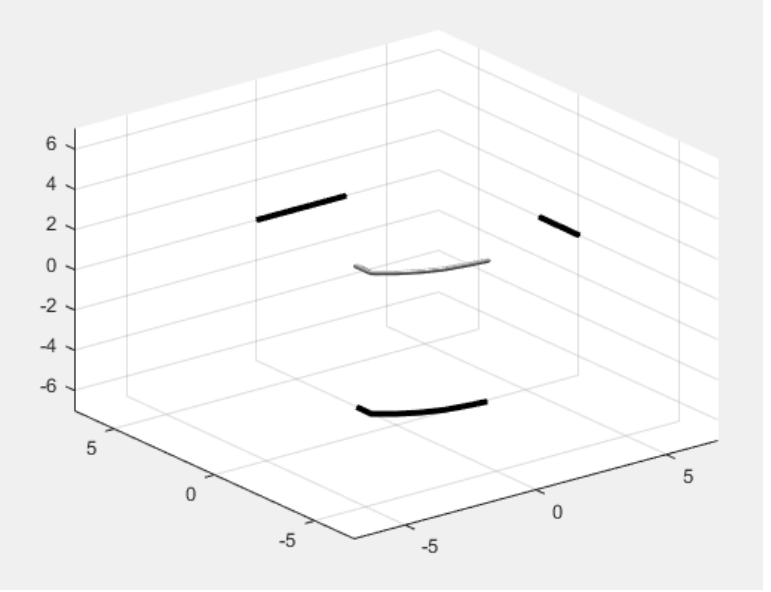

Figure 5.6: Case B: Snake position at end of simulation.

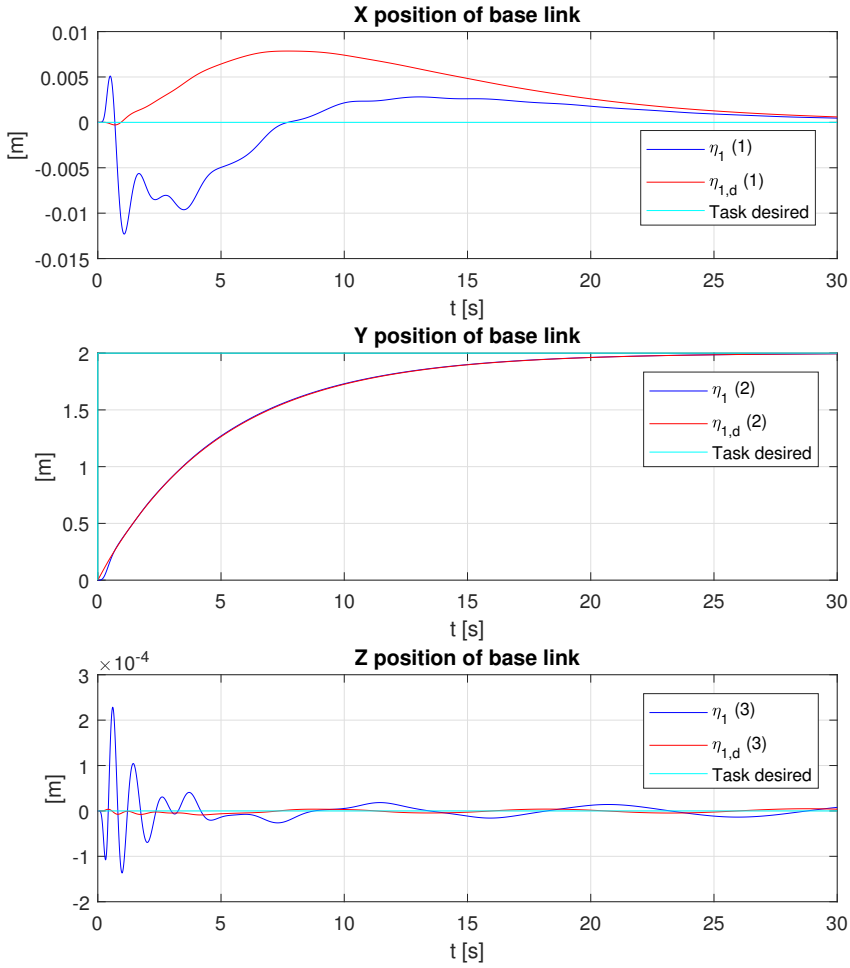

Figure 5.7: Case B: Base link position in inertial frame.

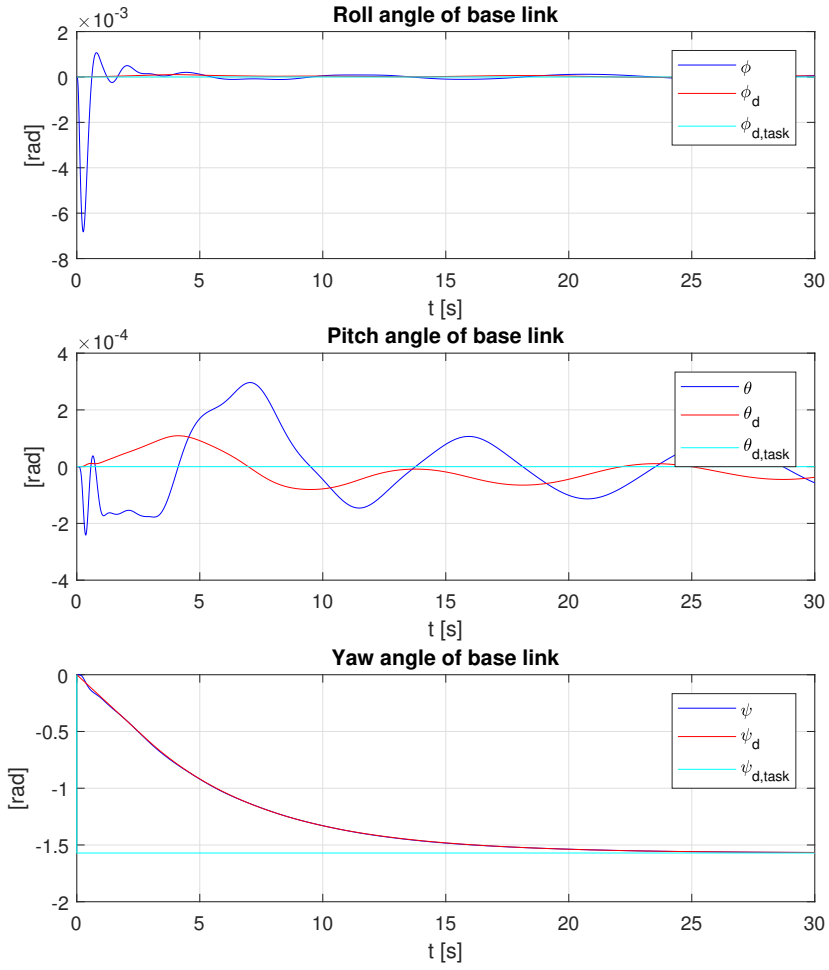

Figure 5.8: Case B: Base link orientation with reference to inertial frame.

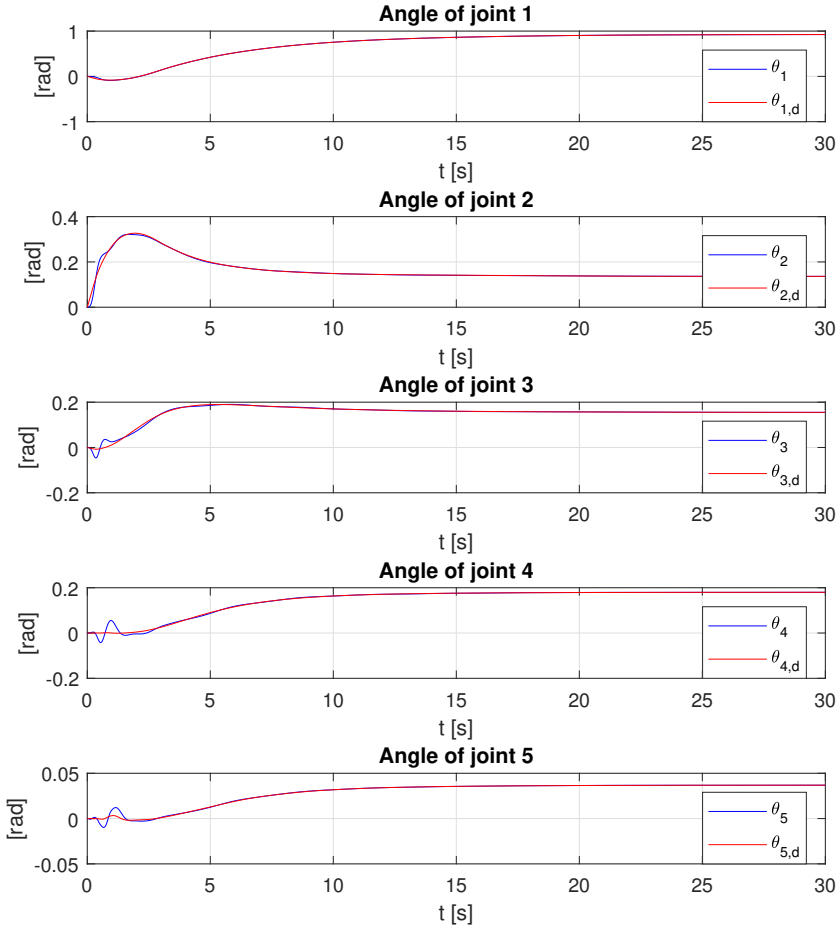

Figure 5.9: Case B: Joint angles.

5.3 Case C

The third case to be simulated for the implemented adaptive controller and inverse kinematics, shall perform three prioritized tasks. Two of the tasks sets the position and orientation of the end effector and the base link. The last task is the joint-limit task, which holds top priority. This means the snake robot shall achieve the specified end effector and base link configuration, but the joint limit task will control the reference if any of the joints are outside the accepted range. The desired position of the end effector and base link are listed in equations (5.2) and (5.3) while the limit of the joint limit task is set to $\pm\pi/3$. The target position is visualized in figure 5.10. Results from the simulation can be seen in figures 5.12, 5.13, 5.14, 5.15, 5.16 and 5.17. Figure 5.14 shows the end effector position, figure 5.15 shows the end effector orientation, figure 5.16 shows the base link position, figure 5.17 shows the base link orientation, figure 5.12 shows the joint angles and figure 5.13 shows the joint angular velocities. The final achieved position of the snake robot is visualized in figure 5.11.

$$\boldsymbol{\eta}_{ee,task} = [1 \quad 0 \quad 0 \quad 0 \quad 0 \quad \pi/2]^\top \tag{5.2}$$

$$\boldsymbol{\eta}_{base,task} = [-2 \quad 0 \quad 0 \quad 0 \quad 0 \quad -\pi/2]^\top \tag{5.3}$$

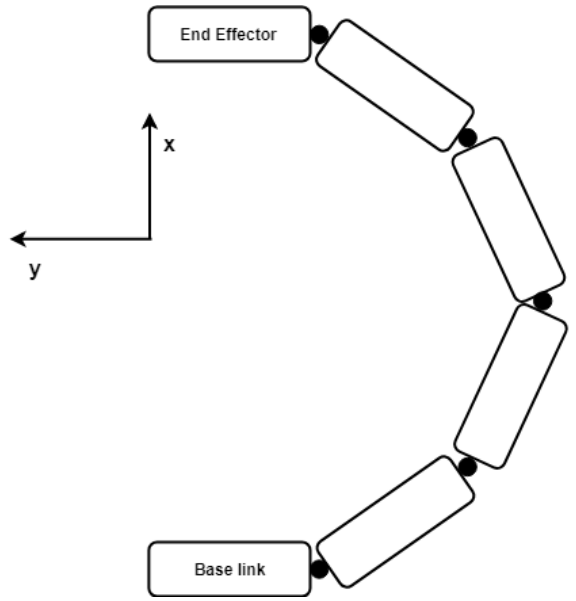

Figure 5.10: Case C: The desired task position of the end effector and base link shown from above.

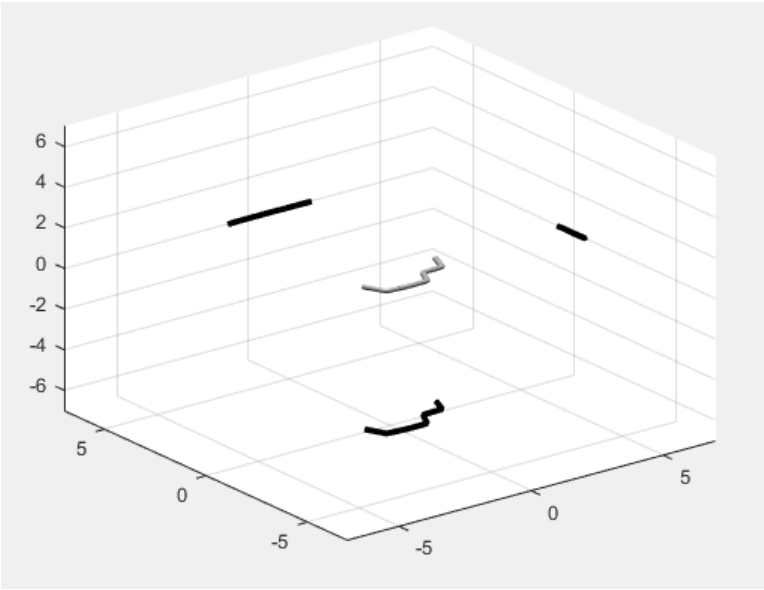

Figure 5.11: Case C: Snake position at end of simulation.

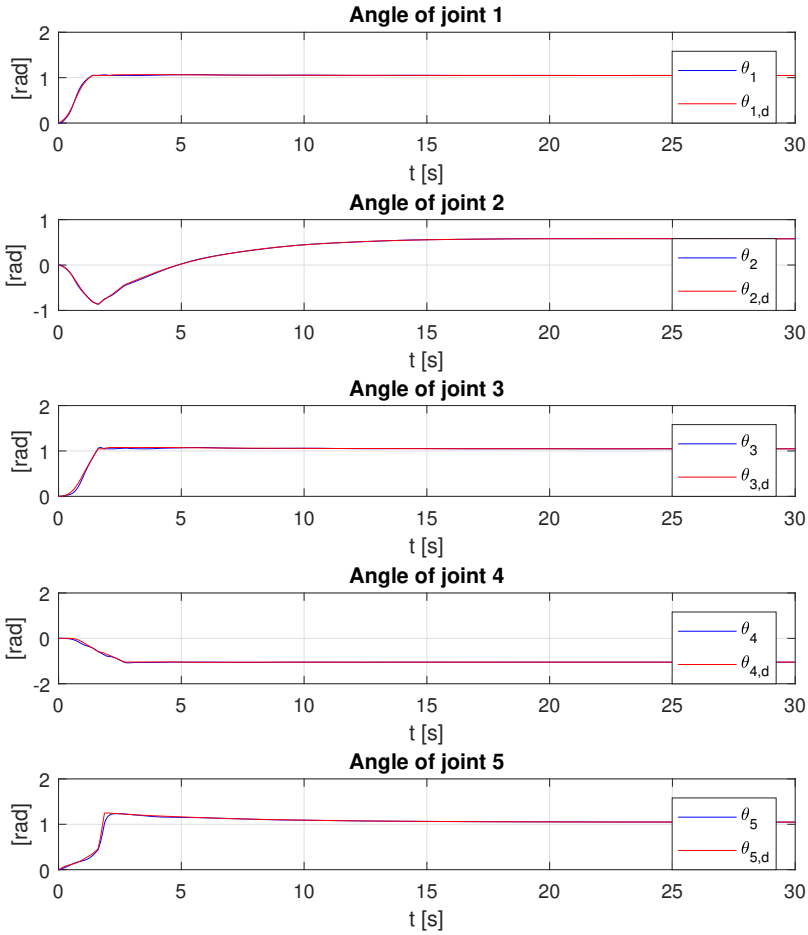

Figure 5.12: Case C: Joint angles.

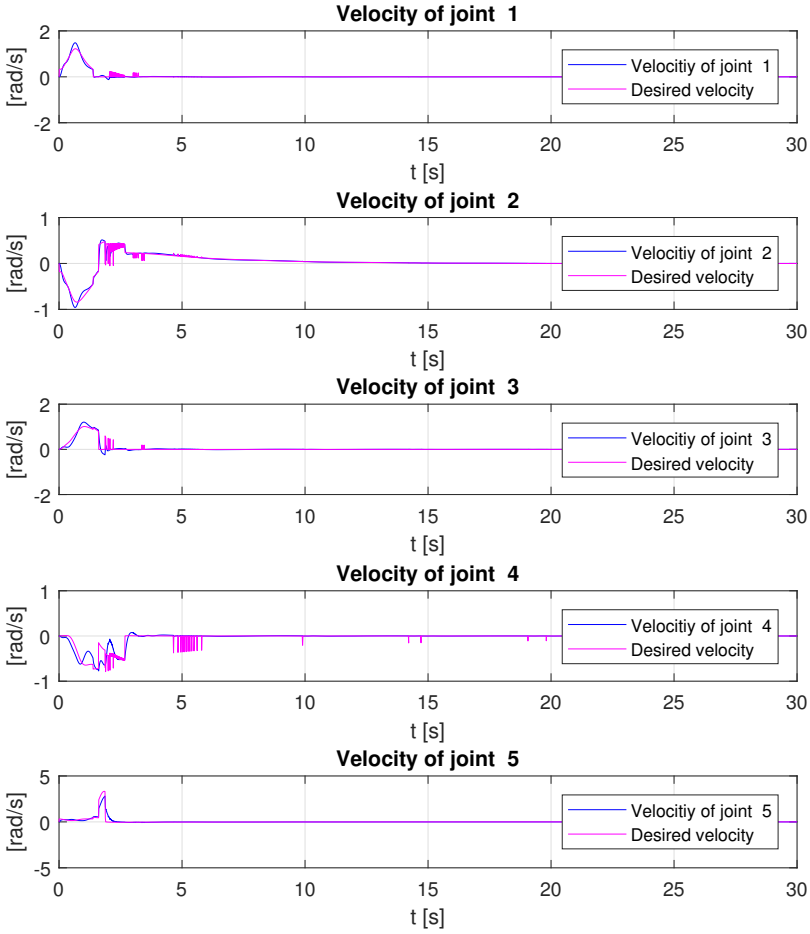

Figure 5.13: Case C: Joint angular velocities.

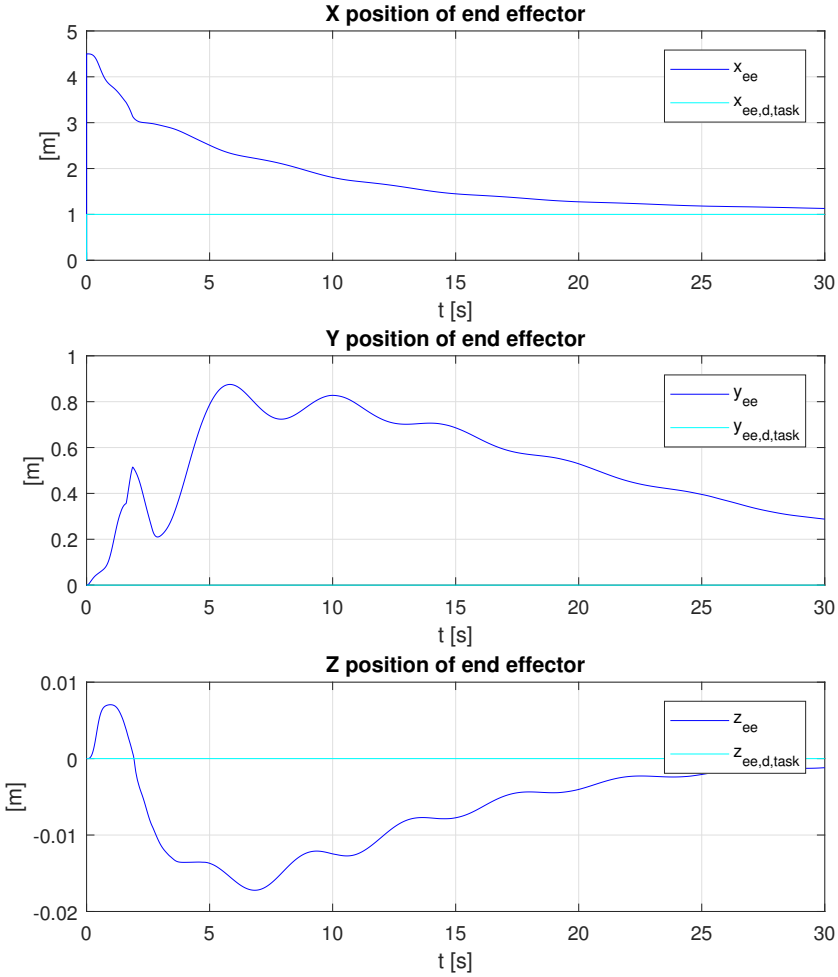

Figure 5.14: Case C: End effector position.

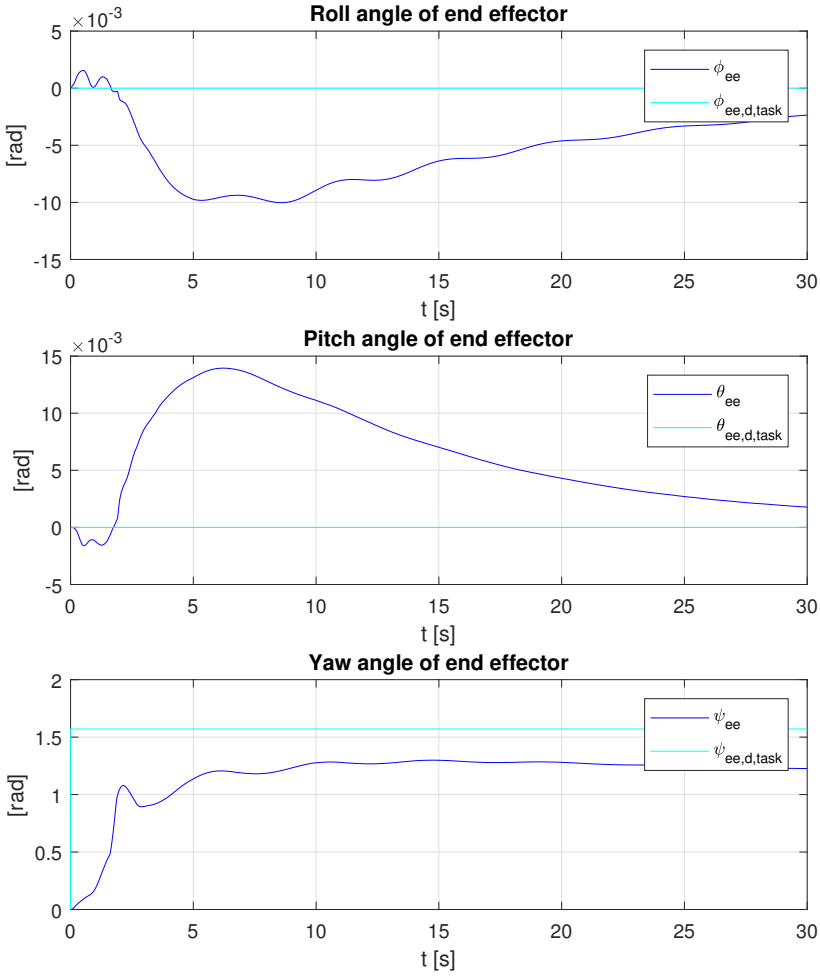

Figure 5.15: Case C: End effector orientation.

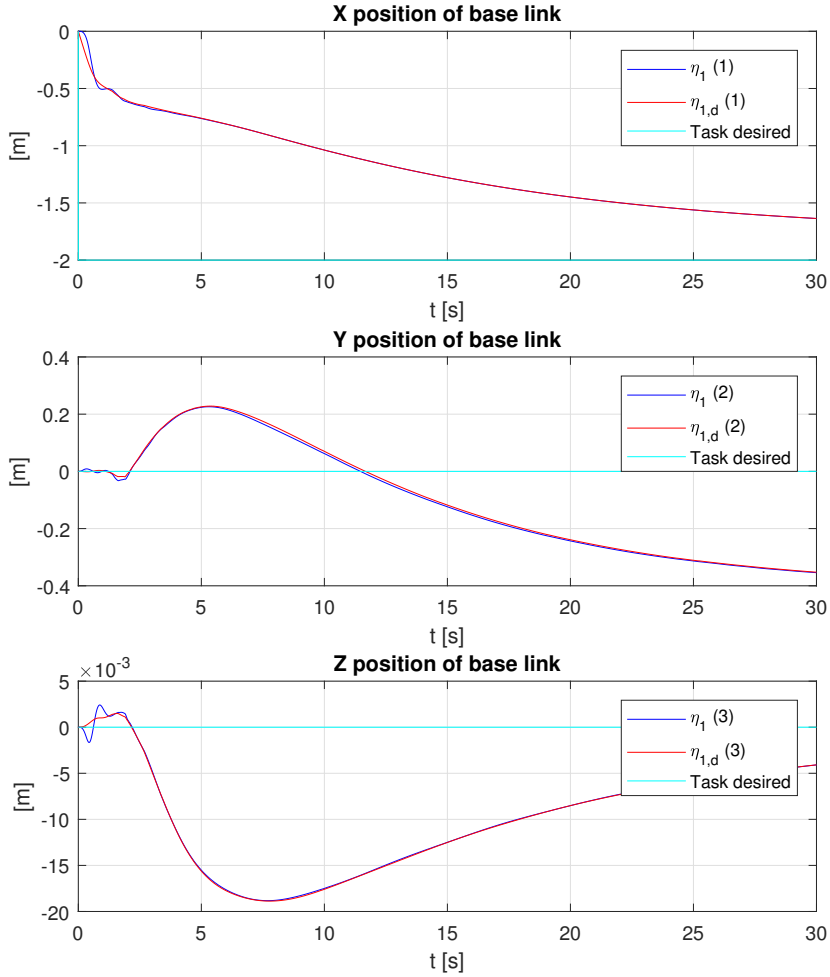

Figure 5.16: Case C: Base link position.

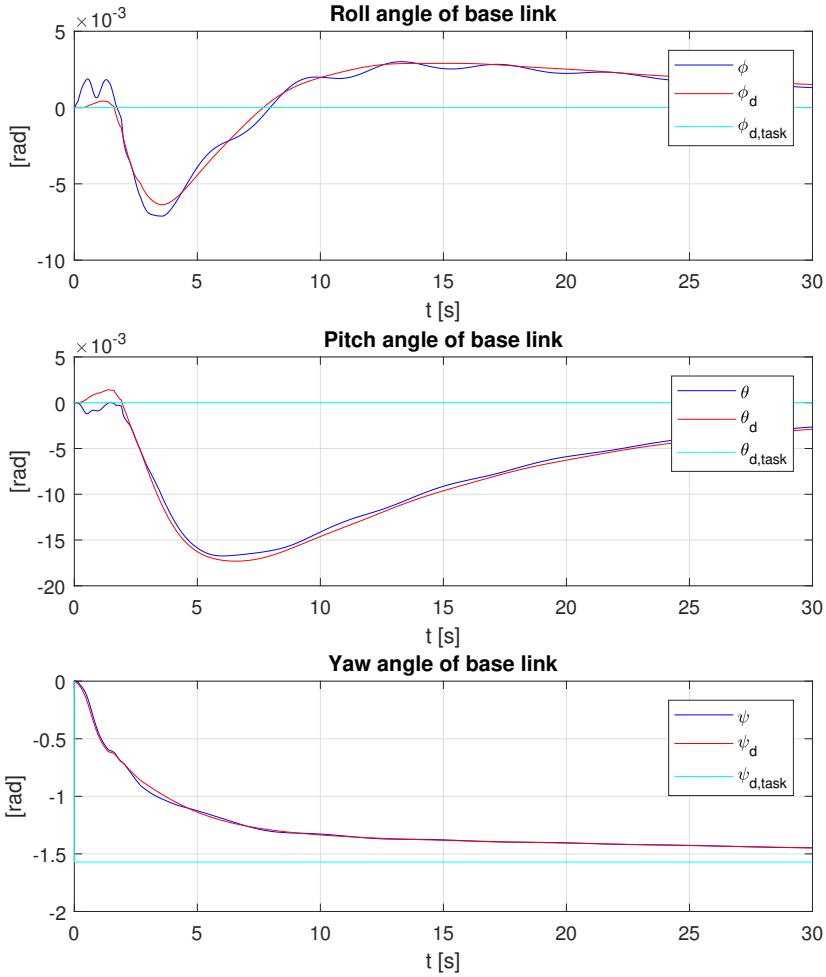

Figure 5.17: Case C: Base link orientation.

5.4 Case D

Case D simulates the scenario that the snake robot will pick up an object. As the snake might have grip arms at each end of its body, the object could be picked up with either end. For this simulated case, the object is assumed to be picked up by the base link grip arm. Although the simulator does not include any grip arms or objects other than the snake robot itself, the case will be simulated by adding weight to the base link once it is within reach of the point where the object is supposed to be. The only task that needs to be interpreted for this simulation is the base link configuration task. Once the base link is within 0.2 meters of the desired point, a theoretical weight of 60 [N] is put on it. The base link task position is chosen as in equation (5.4). Figures 5.19, 5.20, 5.21 and 5.22 shows the plotted values for the base position and orientation, and the base thrust and torque given by the controller.

$$\boldsymbol{\eta}_{base,task} = \begin{bmatrix} -2 & 3 & 0 & 0 & 0 & 0 \end{bmatrix}^\top \tag{5.4}$$

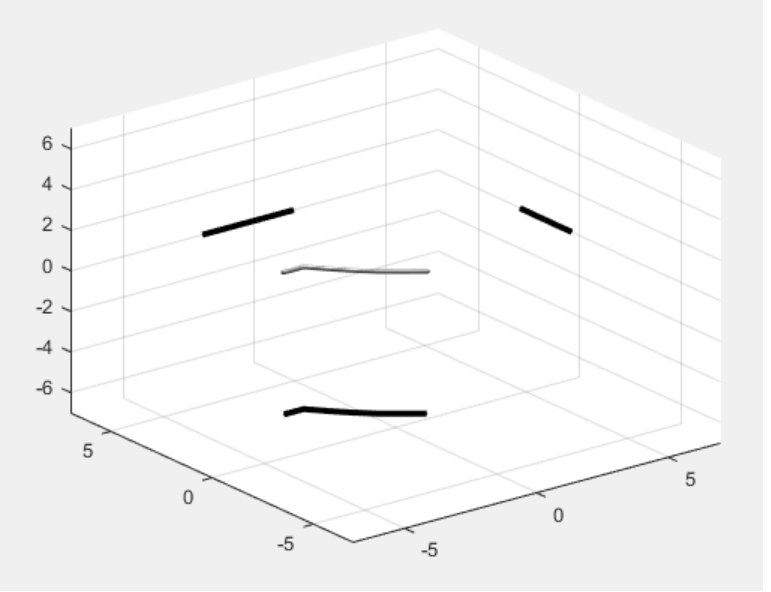

Figure 5.18: Case D: Snake position at end of simulation.

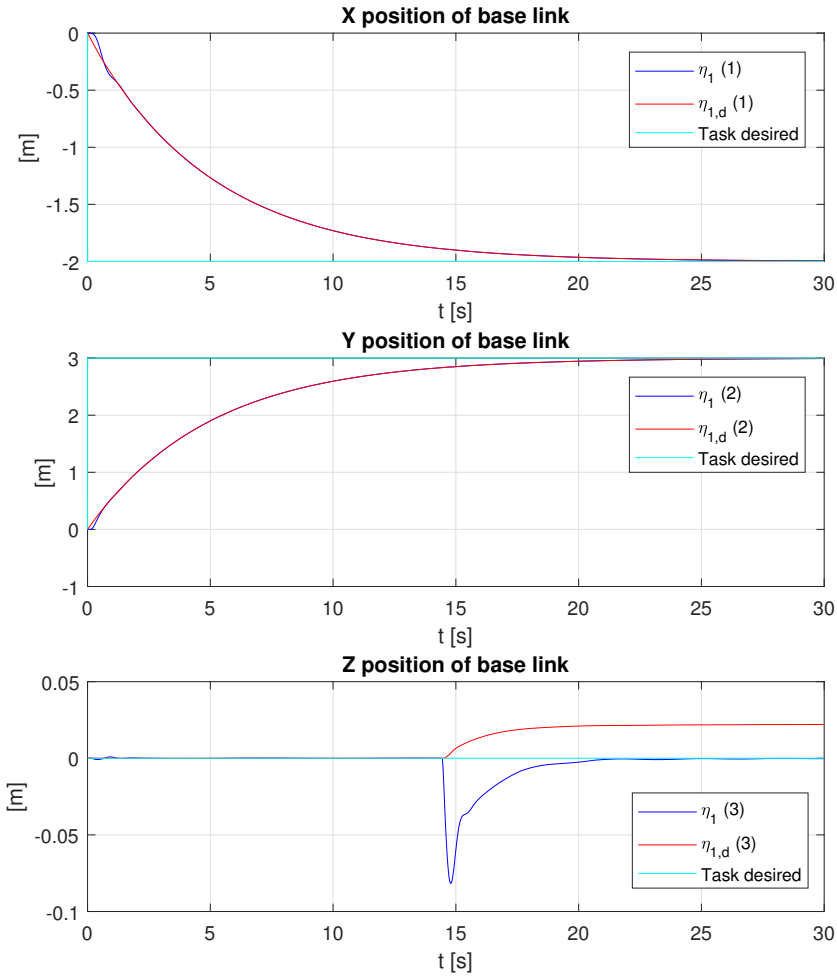

Figure 5.19: Case D: Base link position.

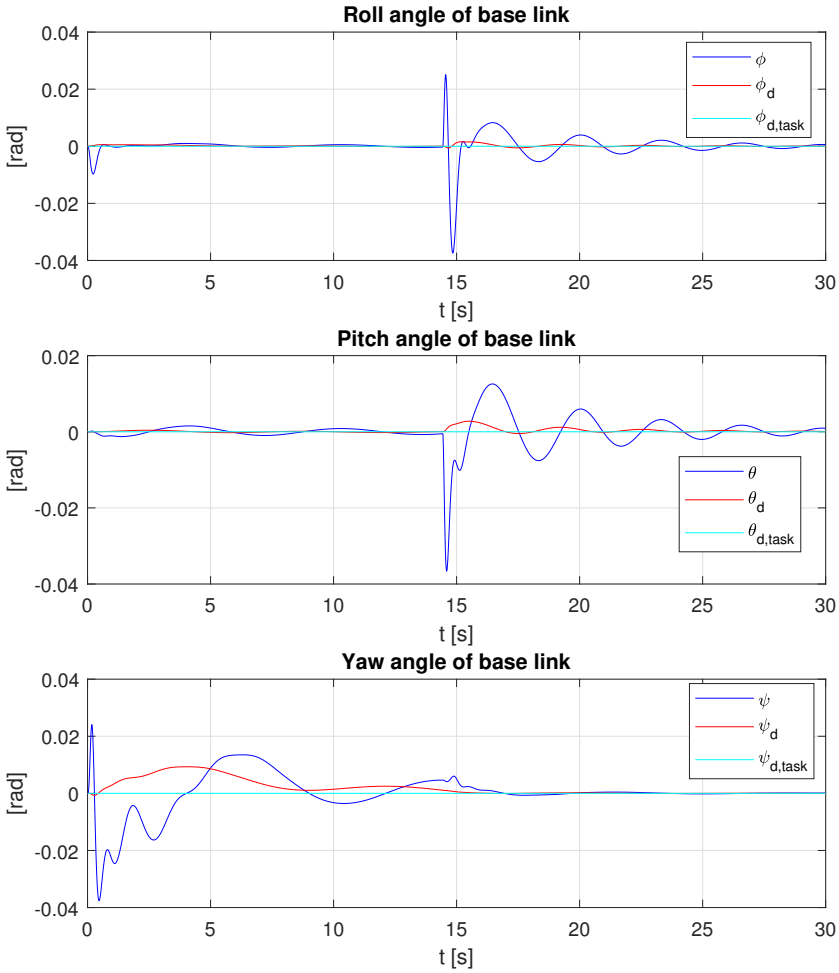

Figure 5.20: Case D: Base link orientation.

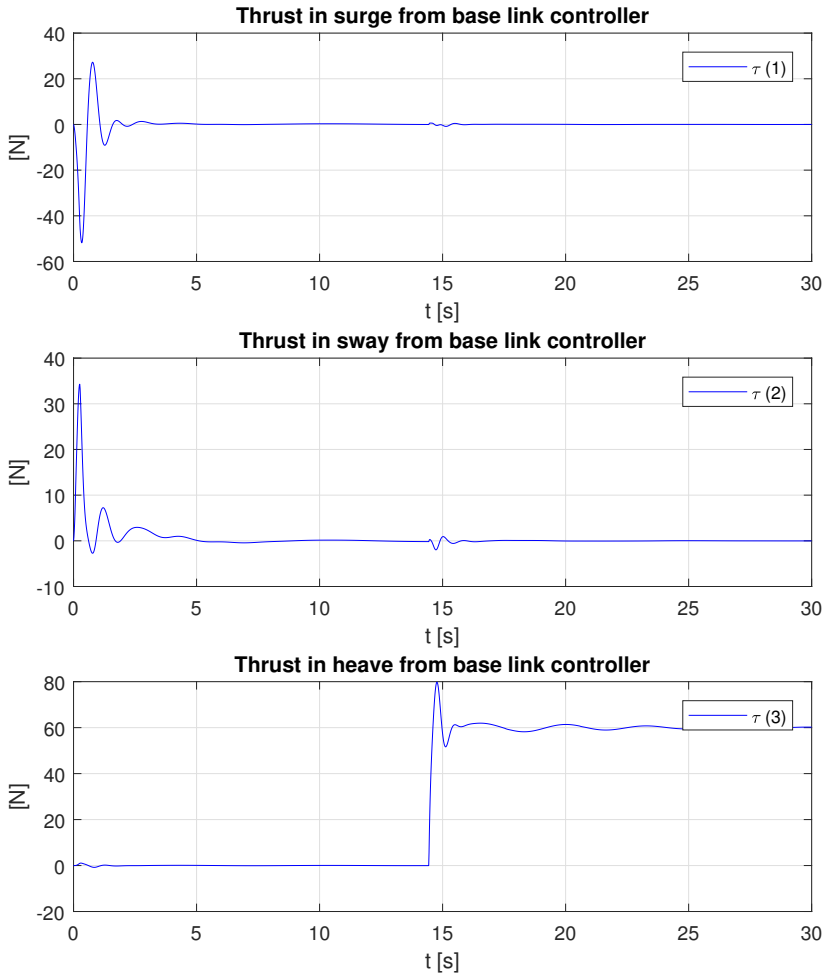

Figure 5.21: Case D: Thrust on base link from controller.

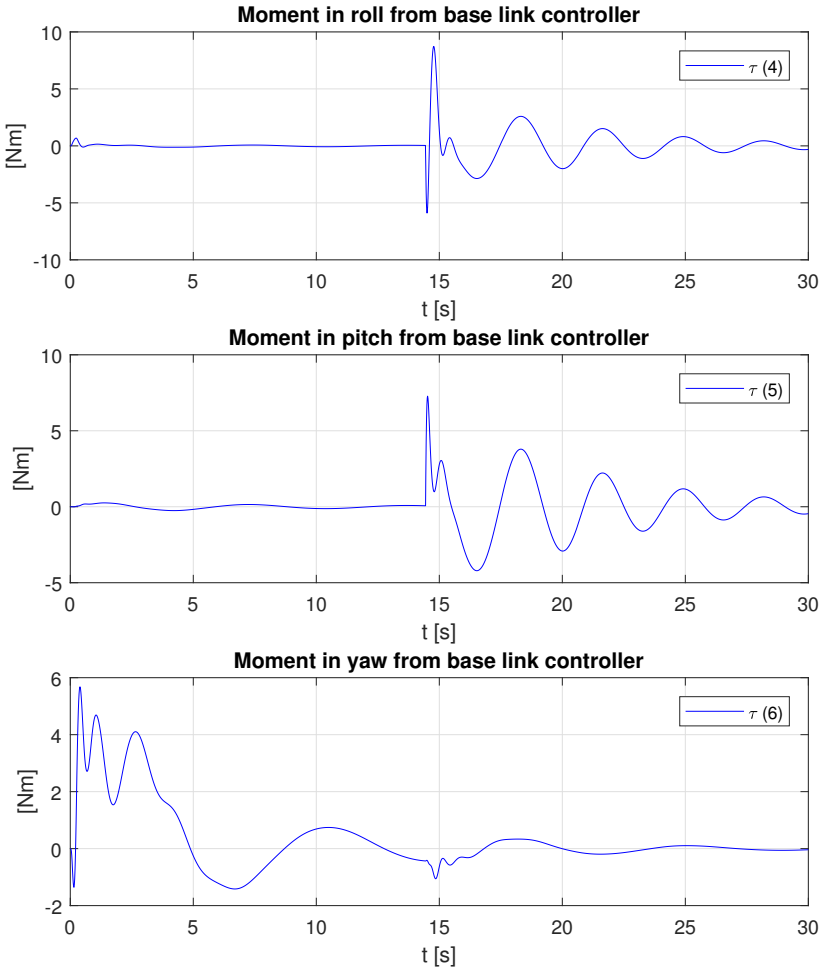

Figure 5.22: Case D: Torque on base link from controller.

Chapter 6

Discussion

All the different cases that have been simulated, tested various aspects of the implemented adaptive controllers and inverse kinematics. The first interesting results comes from case A and case B. In the two cases the system was set to do the same task, except case B used the weighted pseudo inverse. This would seem like a small difference, but the two cases displays different results on how the task was solved. Nevertheless, both of the simulated systems were able to achieve the desired task of placing the base link at a certain point, with a certain orientation. The inverse kinematics of case A chose to keep zero angle in the joints, even though this requires much more movement from the snake robot. Therefore, the whole snake turned to the side. For case B, the inverse kinematics algorithm tried to move the snake as little as possible. This is seen from the plots of the joint angles from the two cases, figure 5.5 and figure 5.9. The reference signal for the joint angles of case A does not move away from zero at all, while the reference signal for case B allows the joints to move away from zero. The difference can be explained from the two functions that the different pseudo inverses aims to minimize (3.43) (3.45). Where case A tries to minimize the movement of the snake robot's degrees of freedom, case B tries to minimize the kinetic energy of the snake robot.

The third case that was simulated was Case C, section 5.3. For this case the inverse kinematics and controllers were not able to achieve the desired position or orientation of either the end effector- or the base link task. That is because of the top priority task for this case, which was the joint limit task. As seen in figure 5.12, the joint angles are not exceeding $\pm\pi/3$, which is the limit set by the top prioritized task. The limit ensures that the joint angles are put back within the limits if they are outside the accepted range. In figure 5.13 the joint velocities and joint reference velocities are shown. It can clearly be seen that the reference signals has spikes early in the simulation. These spikes happen when the joint limit task takes control of the reference. Multiple spikes in a row are a result of the joint angles jumping back and forth over the joint angle limit. However, it is interesting that the reference keeps pushing the joints towards the limit when the second joint still has not reached its limit. The explanation might come from the fact that the states of the snake robot influences each other. Maybe if the second joint is pushed further,

it makes another joint go over the limit and therefore is being pushed back.

On the other hand, the simulation of case C was successful in one aspect. The joint limit task was in fact able to keep the joint within the accepted range. As the joint limit task held top priority, less successful completion of lower prioritized tasks must be accepted. It might be argued that the system should in fact be able to complete all three tasks. From figures 5.11 and 5.12 it is seen that if joint number 4 would have turned to a positive angle instead of negative, the other joints would not have reached their limit and all tasks could have been achieved. However, the tasks for the end effector and base link configuration does not consider the joint limit task at all. These two lower prioritized tasks will simply make the inverse kinematics give a reference as if there was no other task to consider. The fourth joint will get a negative angle reference, as this command would help the snake robot reach the desired end effector configuration if there was no other task preventing it from moving as far as it wants. To avoid such drawbacks, the inverse kinematics would have to be implemented in some way that allows for finding a solution, given specific limitations to the system.

The last case, case D, aimed to reach a given position and orientation for the base link. When the position was reached, a weight was put on it to simulate a more applicable case where the snake robot picks up an object. From the plots of the simulation it is seen that the weight was put on right before 15 seconds of the simulation. At the beginning of the simulation the inverse kinematics and the adaptive controllers are able to bring the base link to the desired position, as seen in figures 5.19 and 5.20. Once it is within reach of the object, the weight is put on, which clearly shows for the Z position of the base link, figure 5.19. When the weight is applied, the base link naturally starts to move downwards a bit before the controller is able to bring it back up to the desired position again. Also, the pitch and roll angles of the base link, figure 5.20, experiences some deviance from the desired angle because of the weight. The roll angle and the pitch angle are brought back to the position they were before the weight was put on, but there are some oscillations in the two states before getting there. Although the height of the oscillations are very small, they could arguably have been avoided if the controller had been tuned less aggressive. At the same time, the states of all the simulations had some deviance from the references at the beginning. That is only normal, as the controllers have to adapt and "get used" to the system they are controlling. As the systems dynamics changes when the weight is applied, it seems reasonable that the controllers need a bit of time to adjust to this. However, the oscillations in the roll and pitch angles lasts for a much longer time than the oscillations at the start of the simulation, which is not preferable. The base controller output of figures 5.21 and 5.22 fits the behaviour of the base link that have been discussed. Although the oscillations are not particularly fast, such oscillations are not desirable for å long period.

Another interesting aspect from the results of case D can also be seen in figure 5.19. The reference signal for the Z position of the base link starts to grow as the weight is put on. That would be normal as the inverse kinematics tries to pull the base link up again. However, as the Z position of the base link comes back to the desired position at zero, the reference signal stays positive and does not decrease back to zero. This is unlike all the other results where the controller has been tracking the reference without offset. The big difference is of course the weight. For none of the simulated cases has there been a constant force pushing the snake robot out of position. The fact that the base controller is

not able to follow the reference signal and stops with a constant offset means that it has some struggles for constant external forces. The implemented controller has no typical integral term, although the third term in the control law (3.26) should be able to remove large offsets. The $\mathbf{K}_i$ matrices are calculated with the update parameter which integrates the norm of the error. But as mentioned, the results form case D shows that the controller implemented for this case is not able to remove the small steady state error. This exact drawback of the controller was actually pointed to in (Lee & Yuh 1999), where a small integral term was added to the adaptive controller.

Although most of the results shows that the implemented kinematic control approach works, there are some issues that have not been pointed to. The control problem was split into two parts, the base controller and the joint controller. Where the joint controller seems like a very reasonable way to actually control the joints, the implementation of the base controller might have some possible improvements. The choice of only controlling the movement of the snake through its base link is obviously not the best approach for all scenarios. It works very well if the snake should move backwards with zero angle in all joints, but if it were to move forwards, the drag effects would press the joints to get some angular offset and give the whole snake unwanted rotation. Of course, the joint controller would try to get rid of this offset, but it would be easier in such a case to move the snake in terms of its center of gravity, or even further to the front of the snake robot. Including some sort of thrust allocation, where the thrusters work at different links could also improve on the performance in such a case.

Chapter 7

Conclusion

The overall aim for the thesis was to implement and test an adaptive control algorithm together with an inverse kinematics algorithm. To do so, a literature review on the relevant topics was first provided. Relevant theory needed to work with the simulation model was then presented. The process of implementing the theory for a snake robot was then described, before the results from simulating the system for four different case studies were presented. The results were discussed in detail.

Most of the results showed good performances from the system, where it was able reach the defined tasks, and the controllers were able to track the references satisfactory. There were also a couple of things that could have been better. The references generated by the inverse kinematics in case C were not able to achieve the second and third prioritized tasks. This failure was due to the joint limit tasks which had top priority for the case. From the joint limit perspective, the inverse kinematics was successful as the other tasks did not interfere with the top priority task. However, there was a possible solution to achieve all three tasks. Ideally, the inverse kinematics should have been able to find this solution.

The other unsatisfactory observation from the results was that the controller for case D had a small offset compared to the reference when extra weight was added to the snake robot. The adaptive controller should be able to work good under such changes and still follow the reference. As noted in the discussion, a small integral term could help avoid such small stationary offsets.

With the results in mind, it is concluded that the adaptive controller and inverse kinematics works well for simplest cases. The controller follows the reference satisfactory for almost all cases, but the inverse kinematics might not always find the best solution to achieve several prioritized tasks.

7.1 Further Work

The kinematic control implemented had some drawbacks which was shown in the results. For further work of this approach to control snake robots, defining the base controller differently could be useful. The next obvious step to take would be to also implement thrust allocation to see whether the presented system could do well for an actual snake robot, not only on the simulator. Other approaches on implementing the inverse kinematics could also be useful. Maybe an iterative method of some sort, that would be able to find several solutions and at the same time take care of joint constraints or any other constraints that might be present. This could also be extended to external things, for example avoiding objects. For the part concerning tasks, it could also be investigated if a block able to switch between tasks and prioritize them differently for different scenarios, could benefit the performance.

Bibliography

Antonelli, G. (2013), *Underwater Robots*, 3rd edn, Springer Publishing Company, Incorporated.

Antonelli, G., Caccavle, F., Chiaverini, S. & Fusco, G. (2001), A novel adaptive control law for autonomous underwater vehicles, *in* 'Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)', Vol. 1, IEEE, pp. 447–452 vol.1.

Choi, Y. (2008), 'Singularity-robust inverse kinematics using lagrange multiplier for redundant manipulators', *Journal of Dynamic Systems, Measurement, and Control* **130**(5).

Erleben, K. & Andrews, S. (2019), 'Solving inverse kinematics using exact hessian matrices', *Computers Graphics* **78**, 1 – 11.
URL: *http://www.sciencedirect.com/science/article/pii/S0097849318301730*

Fossen, T. (2011), *Handbook of Marine Craft Hydrodynamics and Motion Control*.

From, P. J., Gravdahl, J. T. & Pettersen, K. Y. (2014), *Vehicle-Manipulator Systems: Modeling for Simulation, Analysis, and Control*, Advances in Industrial Control, 2014 edn, Springer London, London.

Ho, E. S. L., Komura, T. & Lau, R. W. H. (2005), Computing inverse kinematics with linear programming, *in* 'Proceedings of the ACM Symposium on Virtual Reality Software and Technology', VRST '05, ACM, New York, NY, USA, pp. 163–166.
URL: *http://doi.acm.org/10.1145/1101616.1101651*

Lee, P.-M. & Yuh, J. (1999), Application of non-regressor based adaptive control to an underwater mobile platform-mounted manipulator, *in* 'Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No.99CH36328)', Vol. 2, pp. 1135–1140 vol. 2.

Liljebäck, P., Pettersen, K., Stavdahl, & Gravdahl, J. (2012), 'A review on modelling, implementation, and control of snake robots', *Robotics and Autonomous Systems* **60**(1), 29 – 40.
URL: *http://www.sciencedirect.com/science/article/pii/S0921889011001618*

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

Minamide, N., Nikiforuk, P. & Gupta, M. (1984), 'Design of adaptive pole-placement controllers for plants of unknown order', *Systems and Control Letters* **4**(6), 347–358.

Moradi, H. & Lee, S. (2005), 'Joint limit analysis and elbow movement minimization for redundant manipulators using closed form method', *Advances In Intelligent Computing, Pt 2, Proceedings* **3645**, 423–432.

Morgansen, K. A., Duidam, V., Mason, R. J., Burdick, J. W. & Murray, R. M. (2001), Nonlinear control methods for planar carangiform robot fish locomotion, *in* 'Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)', Vol. 1, pp. 427–434 vol.1.

Morgansen, K. A., Triplett, B. I. & Klein, D. J. (2007), 'Geometric methods for modeling and control of free-swimming fin-actuated underwater vehicles', *IEEE Transactions on Robotics* **23**(6), 1184–1199.

Pettersen, K. Y. (2017), 'Snake robots', *Annual Reviews in Control* **44**, 19 – 44.
URL: *http://www.sciencedirect.com/science/article/pii/S1367578817301050*

Rollinson, D. & Choset, H. (2013), Gait-based compliant control for snake robots, IEEE, pp. 5138–5143.

Sastry, S. & Bodson, M. (1989), *Adaptive Control: Stability, Convergence, and Robustness*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

Shugen Ma, H., Araya, H. & Li Li, H. (2001), Development of a creeping snake-robot, Vol. 2001-, IEEE, USA, pp. 77–82.

Tanaka, M., Kon, K. & Tanaka, K. (2015), 'Range-sensor-based semiautonomous whole-body collision avoidance of a snake robot', *IEEE Transactions on Control Systems Technology* **23**(5), 1927–1934.

Tanaka, M. & Tanaka, K. (2017), 'Shape control of a snake robot with joint limit and self-collision avoidance', *IEEE Transactions on Control Systems Technology* **25**(4), 1441–1448.

Wiriyacharoensunthorn, P. & Laowattana, S. (2002), Analysis and design of a multi-link mobile robot (serpentine), *in* '2002 IEEE International Conference on Industrial Technology, 2002. IEEE ICIT '02.', Vol. 2, pp. 694–699 vol.2.

Worst, R. & Linnemann, R. (1996), Construction and operation of a snake-like robot, *in* 'Proceedings IEEE International Joint Symposia on Intelligence and Systems', pp. 164–169.

Ye, C., Ma, S., Li, B. & Wang, Y. (2004), Locomotion control of a novel snake-like robot, *in* '2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)', Vol. 1, pp. 925–930.

Yuh, J. (1996), 'An adaptive and learning control system for underwater robots', *IFAC Proceedings Volumes* **29**(1), 145 – 150. 13th World Congress of IFAC, 1996, San Francisco USA, 30 June - 5 July.
URL: *http://www.sciencedirect.com/science/article/pii/S1474667017576533*

Yuh, J., Nie, J. & Lee, C. (1999), 'Experimental study on adaptive control of underwater robots', *Proceedings - IEEE International Conference on Robotics and Automation* **1**, 393–398.

Zhang, A., Ma, S., Li, B., Wang, M., Guo, X. & Wang, Y. (2016), 'Adaptive controller design for underwater snake robot with unmatched uncertainties', *Science China Information Sciences* **59**(5), 052205.
URL: *https://doi.org/10.1007/s11432-015-5421-8*

Zuo, Z., Wang, Z., Li, B. & Ma, S. (2009), Serpentine locomotion of a snake-like robot in water environment, *in* '2008 IEEE International Conference on Robotics and Biomimetics', pp. 25–30.

Appendix: MATLAB code

USRsim2.m

```
%This is a simple smiulator for an underwater snake-like robot. The code is
%not optimized for speed and there may still be bugs. Please report back to
%me if you find anything fishy :)
%
%Henrik

%clc;
clear
%close all

%% Define Physical Parameters
pp.denw = 1000;      %density of water
pp.grav = 9.81;      %gravitational acceleration
pp.velc = [0 0 0]'; %constant irrotational current velocity, inertial frame
pp.gdir = [0 0 -1]';%direction of gravity in inertial frame

%% Adding switch to put on weight when base link reaches target.
pp.switch_weight = 0;

%% Define Snake Parameters
snake.link_vec  = [8 8 8 8 8 8]; %The type of links of the snake
snake.joint_vec = [3 3 3 3 3];   %The type of joints of the snake
snake.n         = length(snake.link_vec);

C_a_C   =   1; % Added mass coefficient for the cross section,
               %   1 is the theoretical inviscid result
C_d_1   =  .1; % nonlinear drag coefficient in surge
```

```
28 C_d_4    =   .1; % nonlinear drag coefficient in the roll
       direction for
29                 %   the cross-section
30 C_d_C    =   .8; % nonlinear crossflow drag coefficient
31 C_d_L    =   .1; % Linear cross-sectional drag coefficient
32 alpha    =   .1; % Added mass ratio in surge/heave for a link
33 beta     =   .1; % Linear drag parameter in surge
34 gamma    =   .1; % Linear drag parameter in roll
35
36 masscoeff = [C_a_C, alpha];
37 dragcoeff = [C_d_1, C_d_4, C_d_C, C_d_L, beta, gamma];
38 clear C_a_C C_d_C C_d_1 C_d_4 C_d_L alpha beta gamma
39
40 %% Define Link Properties
41 link = define_link_properties_eely(pp, masscoeff, dragcoeff
       );
42
43 %% Define Joint Properties
44 joint = define_joint_properties();
45
46 %% Define Simulation Parameters
47 Tmax = 1;
48 h = 0.001;                      %RK4 step size
49 deltaT_interp = 0.1;            %Time step size for the
       interpolated solution
50 deltaT_plot = 0.1;              %Time step size for the
       plotted solution
51
52 x_0 = zeros(26+3*snake.n,1); %Initial conditions
53 x_0(4) = 1;                     %"zero" quaternion
54 x_0(8:6+snake.n) = 0;           %Initial angle of joints
55
56 %% Build Snake
57 snake = build_snake(snake, link, joint, pp);
58
59 %% Simulate
60 output = RK4solver(h,Tmax,x_0,snake,pp);
61 T_log = output.T_log;
62 x_log = output.x_log;
63
64 %% Plot results
65 % run plotResults.m;
66 %% Show Movie
67 % run showMovie.m;
```

RK4solver.m

```
function output = RK4solver(h,Tmax,x_0,snake,pp)
% RK4 Function to go through the simulation step for step
    with Runge Kutta
% 4th order method.


%% Number of steps
n_steps = Tmax/h;
T_log = linspace(0,Tmax,n_steps+1);

%% Preparing output
x_log = zeros(26+3*snake.n,n_steps+1);
tau_d_log = zeros(6,n_steps+1);
joint_torque_log = zeros(snake.n-1,n_steps+1);
x_ref_log = zeros(20+4*snake.n,n_steps+1);
orientation_log = zeros(3,n_steps+1);
eta_ee_log = zeros(6,n_steps+1);
% eta_ee_ref_log = zeros(6,n_steps+1);
theta_dotd_log = zeros(snake.n-1,n_steps+1);
eta_ee_task_log = zeros(6,n_steps+1);
eta_base_task_log = zeros(6,n_steps+1);
nominal_task_log = zeros(snake.n-1,n_steps+1);

joint_controller_log.phi_1 = zeros(snake.n-1,n_steps+1);
joint_controller_log.phi_2 = zeros(snake.n-1,n_steps+1);
joint_controller_log.phi_3 = zeros(snake.n-1,n_steps+1);
joint_controller_log.phi_4 = zeros(snake.n-1,n_steps+1);
joint_controller_log.phi_5 = zeros(snake.n-1,n_steps+1);


%% Initial state
x_log(:,1) = x_0;

%% Simulation
for i = 1:n_steps
    x = x_log(:,i);

    %% Get states
    pos         =  x(1:3);
    quat        =  x(4:7);
    theta       =  x(8 : 6+snake.n);
    zeta        =  x(7+snake.n : 11+2*snake.n);
    nu          =  zeta(1:6);

```

```
44      %Parameters from updating factor, gamma, in the
            adaptive
45      %control law for the whole body and for joints.
46      gamma_hat = x(11+2*snake.n+1 : 11+2*snake.n+5);
47      gamma_hat_link = x(11+2*snake.n+6 : 11+2*snake.n+10);
48
49      %Desired positions for base link and joints
50      eta_d = x(12+2*snake.n+10 : 17+2*snake.n + 10);
51      theta_d = x(18+2*snake.n+10 : 16+3*snake.n + 10);
52
53      %% Compute rotation matrices and jacobians
54      [R, J]     =  comp_rot_jac(quat,theta,snake);
55
56      %% Test path
57  %     time = i*h;
58      %[zeta_ref, eta_ee] = generateTestPath(snake,x,R,time);
59
60      %% Find transformation matrix
61      eulerAngles = getEulerAngles(R(:,:,1));
62      T = getTransformationMatrix(eulerAngles);
63      %% Solve inverse kinematics
64      [zeta_ref, eta_ee, logTasks, pp.switch_weight] ...
65          = inverseKinematics(snake,x,R,J,pp.switch_weight);
66
67      zeta_ref_ine = zeta_ref;
68      zeta_ref_ine(1:3) = R(:,:,1)*zeta_ref(1:3);
69      zeta_ref_ine(4:6) = T*zeta_ref(4:6);
70      zeta_ref_ine_prev = x_ref_log(16+3*snake.n:20+4*snake.n
            ,i);
71
72      acc_ref    = calcAcceleration(zeta_ref_ine_prev,
            zeta_ref_ine,h);
73      %acc_ref = zeros(length(zeta_ref),1);
74      eta_dotdotd = zeros(6,1);%acc_ref(1:6);
75      theta_dotdotd = acc_ref(7:5+snake.n);
76
77      %% Transforming to inertial frame, enabling integration
78      eta = [pos; eulerAngles];
79      eta_dot = zeros(6,1);
80      eta_dot(1:3) = R(:,:,1)*nu(1:3);
81      eta_dot(4:6) = T*nu(4:6);
82      velocity_ine_d = R(:,:,1)*zeta_ref(1:3);
83      angvelocity_ine_d = T*zeta_ref(4:6);
84      theta_dotd = zeta_ref(7:5+snake.n);
85
```

```
86      %% Non-regressor based adaptive control
87
88      [tau_d,gamma_hat_dot, log_base_con] =
            AdaptiveControlBase(eta,...
89           eta_dot,eta_d,zeta_ref,eta_dotdotd,gamma_hat,R,T);
90      [joint_torque, gamma_hat_dot_joint, log_joint_con] =
            ...
91           AdaptiveControlJoint(x,snake, theta_d, theta_dotd
                ,...
92           theta_dotdotd, gamma_hat_link);
93
94      update_param.gamma_hat_dot = gamma_hat_dot;
95      update_param.gamma_hat_dot_link = gamma_hat_dot_joint;
96      update_param.velocity_ine_d = velocity_ine_d;
97      update_param.angvelocity_ine_d = angvelocity_ine_d;
98      update_param.theta_dotd = theta_dotd;
99
100     %% Setting thrusters to zero
101     thrust = zeros(snake.n_thruster_tot,1);
102
103     %% Logging values
104     x_ref_log(:,i+1)          = [eta_d;theta_d;zeta_ref;
            acc_ref; ...
105          zeta_ref_ine];
106     tau_d_log(:,i+1)          = tau_d;
107     joint_torque_log(:,i+1) = joint_torque;
108     orientation_log(:,i+1)   = eulerAngles;
109     eta_ee_log(:,i+1)         = eta_ee;
110     eta_ee_task_log(:,i+1)    = logTasks.ee;
111     eta_base_task_log(:,i+1) = logTasks.base;
112     nominal_task_log(:,i+1) = logTasks.nominal;
113     theta_dotd_log(:,i+1)    = theta_dotd;
114
115     joint_controller_log.phi_1(:,i+1) = log_joint_con.
            phi_1_joint;
116     joint_controller_log.phi_2(:,i+1) = log_joint_con.
            phi_2_joint;
117     joint_controller_log.phi_3(:,i+1) = log_joint_con.
            phi_3_joint;
118     joint_controller_log.phi_4(:,i+1) = log_joint_con.
            phi_4_joint;
119     joint_controller_log.phi_5(:,i+1) = log_joint_con.
            phi_5_joint;
120
121
```

```
122     %% Runge-Kutta 4
123     k1 = f_dyn(x,thrust,tau_d,joint_torque,snake,pp,
            update_param);
124
125     x2 = (x+h*k1/2);
126     x2(4:7) = normQ(x2(4:7)); %Normalizing the quaternion
            vector
127     k2 = f_dyn(x2,thrust,tau_d,joint_torque,snake,pp,
            update_param);
128
129     x3 = (x+h*k2/2);
130     x3(4:7) = normQ(x3(4:7)); %Normalizing the quaternion
            vector
131     k3 = f_dyn(x3,thrust,tau_d,joint_torque,snake,pp,
            update_param);
132
133     x4 = (x+h*k3);
134     x4(4:7) = normQ(x4(4:7)); %Normalizing the quaternion
            vector
135     k4 = f_dyn(x4,thrust,tau_d,joint_torque,snake,pp,
            update_param);
136
137     x_final = x + h*(k1 + 2*k2 + 2*k3 + k4)/6;
138     x_final(4:7) = normQ(x_final(4:7)); %Normalizing the
            quaternion vector
139
140     %% Saturating adaptive integrators and angles
141     x_final = saturateStates(x_final,snake);
142
143     x_log(:,i+1) = x_final;
144     disp(['Time elapsed ', num2str(i*h)])
145
146 end
147 %% Organizing all output values in one struct to simplify
148 output.x_log = x_log;
149 output.tau_d_log = tau_d_log;
150 output.joint_torque_log = joint_torque_log;
151 output.x_ref_log = x_ref_log;
152 output.orientation_log = orientation_log;
153 output.T_log = T_log;
154 output.eta_ee_log = eta_ee_log;
155 output.eta_ee_ref_log = eta_ee_task_log;
156 output.eta_base_task_log = eta_base_task_log;
157 output.nominal_task_log = nominal_task_log;
158 output.theta_dotd_log = theta_dotd_log;
```

```
159 output.joint_controller_log = joint_controller_log;
160
161 end
```

inverseKinematics.m

```
function [zeta_ref, eta_ee, log, switch_weight] =...
    inverseKinematics(snake,x,R,J, switch_weight)
% Uses the task priority and method for inverting jacobian
    matrices
% to output the reference velocity for the snake robot

pos = x(1:3);
theta = x(8:6+snake.n); %Joint angles

%Get end effector position and transformation matrix for
    end effector
[eta_ee,eta_n_b_ee] = getEEpos(snake, pos, theta, R);
T = getTransformationMatrix(eta_ee(4:6));

%Get position of snake base (tail)
eulerAngles = getEulerAngles(R(:,:,1));
eta = [pos; eulerAngles];

%%       Tasks
%%      Position and orientation of end effector
eta_ee_d = [1 0 0 0 0 pi/2]';
eta_ee_d_dot = [0 0 0 0 0 0]';
K_ee = diag([0.2 0.2 0.2 0.2 0.2 0.2],0);
w_ee = eta_ee_d_dot + K_ee*(eta_ee_d - eta_ee);

J_ee = [R(:,:,snake.n)*J(1:3,:,snake.n);
        T*J(4:6,:,snake.n)];

% Saturating error term
w_ee1 = w_ee(1:3);
satee1 = abs(w_ee1) > 1;
w_ee1(satee1) = 1*sign(w_ee1(satee1));

w_ee2 = w_ee(4:6);
satee2 = abs(w_ee2) > 0.5;
w_ee2(satee2) = 0.5*sign(w_ee2(satee2));

w_ee(1:3) = w_ee1;
w_ee(4:6) = w_ee2;

%%     Position and orientation of base link
eta_base_d = [-2 3 0 0 0 0]';
eta_base_d_dot = [0 0 0 0 0 0]';
K_base = diag([.2 .2 .2 .2 .2 .2],0);
```

```
43 w_base = eta_base_d_dot + K_base*(eta_base_d - eta);
44
45 J_base = [R(:,:,1) zeros(3,2+snake.n);
46           zeros(3,3) TransAngEul(eulerAngles) zeros(3,snake
                .n-1)];
47
48 J_base_inv = pinv(J_base);
49
50 % Activate switch when base link is close enough to target
       position
51 if (norm(eta_base_d(1:3) - eta(1:3)) < 0.2)
52     switch_weight = 1;
53 end
54
55 % Saturating error term
56 w_base1 = w_base(1:3);
57 satbase1 = abs(w_base1) > 1;
58 w_base1(satbase1) = 1*sign(w_base1(satbase1));
59
60 w_base2 = w_base(4:6);
61 satbase2 = abs(w_base2) > 0.2;
62 w_base2(satbase2) = 0.2*sign(w_base2(satbase2));
63
64 w_base(1:3) = w_base1;
65 w_base(4:6) = w_base2;
66
67 %%              Nominal configuration
68 joint_d = zeros(snake.n-1,1);
69 joint_d_dot = zeros(snake.n-1,1);
70 K_nconf = 0.2;
71 w_nconf = joint_d_dot + K_nconf*(joint_d-theta);
72
73 J_nconf = [zeros(snake.n-1,6),eye(snake.n-1)];
74
75 sat_nconf = abs(w_nconf) > 0.1;
76 w_nconf(sat_nconf) = 0.1*sign(w_nconf(sat_nconf));
77
78 %%              Joint limit task
79 limit = pi/3;
80 J_joint_limit = zeros(snake.n-1,5+snake.n);
81 w_joint_limit = zeros(snake.n-1,1);
82
83 for i=1:(snake.n-1)
84     if (theta(i) > limit)
85         J_joint_limit(i,6+i) = 1;
```

```
86          w_joint_limit(i) = 0.2*(limit - theta(i));
87      elseif (theta(i) < -limit)
88          J_joint_limit(i,6+i) = 1;
89          w_joint_limit(i) = 0.2*(-limit - theta(i));
90      end
91  end
92
93  %% ************************************************
94  %%                Inverse Kinematics
95
96  %% Weighted pseudoinverse
97  %Mass matrix as weight
98  W = comp_mass(J,snake);
99  Wi = pinv(W,.001);
100
101 % J_ee_winv = Wi*J_ee'*pinv(J_ee*Wi*J_ee',.001);
102 J_base_winv = Wi*J_base'*pinv(J_base*Wi*J_base',.001);
103 % J_joint_limit_winv = Wi*J_joint_limit'*pinv(J_joint_limit
        *...
104 %     Wi*J_joint_limit',.001);
105
106 %% 1 task
107 w_1 = w_base;
108
109 J_1_inv = J_base_winv;
110
111 zeta_ref = J_1_inv*w_1;
112
113
114 %% 2 tasks
115 % w_1 = w_ee;
116 % w_2 = w_base;
117 %
118 % J_1 = J_ee;
119 % J_2 = J_base;
120 %
121 % J_1_inv = J_ee_winv;
122 % J_2_inv = J_base_winv;
123 %
124 % N_1 = eye(5+snake.n) - J_1_inv*J_1;
125 %
126 % zeta_ref = J_1_inv*w_1 + N_1*J_2_inv*w_2;
127
128 %% 3 tasks
129 % w_1 = w_joint_limit;
```

```
130 % w_2 = w_ee;
131 % w_3 = w_base;
132 %
133 % J_1 = J_joint_limit;
134 % J_2 = J_ee;
135 % J_3 = J_base;
136 %
137 % J_1_inv = J_joint_limit_winv;
138 % J_2_inv = J_ee_winv;
139 % J_3_inv = J_base_winv;
140 %
141 % N_1 = eye(5+snake.n) - J_1_inv*J_1;
142 %
143 % J_1_2 = [J_1 ; J_2];
144 % J_1_2_inv = pinv(J_1_2);
145 % N_1_2 = eye(5+snake.n) - J_1_2_inv*J_1_2;
146 %
147 % zeta_ref = J_1_inv*w_1 + N_1*J_2_inv*w_2 + N_1_2*J_3_inv*
        w_3;
148
149 %% Logging tasks
150 log.base = eta_base_d;
151 log.ee = eta_ee_d;
152 log.nominal = joint_d;
153
154
155 end
```

AdaptiveControlBase.m

```
function [tau_d, gamma_hat_dot, log] = AdaptiveControlBase(eta,nu,...
    eta_d,zeta_ref,eta_dotdotd,gamma_hat,R,T)

% Uses non-regressor based adaptive control algorithm to calculate the
% thrust and torque to put on the base link.

%% Control constants and variables
sigma = 10;
f = [3 3 3 1 1]';
k = 3*ones(6,1);

zeta_ref_ine = zeros(6,1);
zeta_ref_ine(1:3) = R(:,:,1)*zeta_ref(1:3);
zeta_ref_ine(4:6) = T*zeta_ref(4:6);
eta_dot_1 = R(:,:,1)*nu(1:3);
eta_dot_2 = T*nu(4:6);
eta_dot = [eta_dot_1; eta_dot_2];
eta_tilde = eta_d - eta;
eta_tilde_dot = zeta_ref_ine(1:6) - eta_dot;
s = eta_tilde_dot + sigma*eta_tilde;

epsilon = [1 1 1 0.2 0.2 0.2]';

s = epsilon.*s;

mu = [0.5 0.5 2.0 0.1 0.1]';
varepsilon = [0.05 0.05 0.05 0.02 0.02]';

delta = 10;

phi_1 = eta_dotdotd; %zeros(6,1);
phi_2 = eta_dot;
phi_3 = k;
phi_4 = eta_tilde_dot;
phi_5 = eta_tilde;
K_1 = gamma_hat(1)*s*phi_1'/max((norm(s)*norm(phi_1)),delta);
K_2 = gamma_hat(2)*s*phi_2'/max((norm(s)*norm(phi_2)),delta);
K_3 = gamma_hat(3)*s*phi_3'/max((norm(s)*norm(phi_3)),delta);
K_4 = gamma_hat(4)*s*phi_4'/max((norm(s)*norm(phi_4)),delta
```

```
    );
K_5 = gamma_hat(5)*s*phi_5'/max((norm(s)*norm(phi_5)),delta
    );

tau_d = K_1*phi_1 + K_2*phi_2 + K_3*phi_3 + K_4*phi_4 + K_5
    *phi_5;

%Transforming tau_d to bodyframe
tau_d(1:3) = R(:,:,1)'*tau_d(1:3);
tau_d(4:6) = pinv(T)*tau_d(4:6);

sat = abs(tau_d) > 100;
tau_d(sat) = 100*sign(tau_d(sat));

%% Logging values to be plotted
%Sending values out to be plotted.
log.phi_1 = phi_1;
log.phi_2 = phi_2;
log.phi_3 = phi_3;
log.phi_4 = phi_4;
log.phi_5 = phi_5;

log.K_1 = K_1;
log.K_2 = K_2;
log.K_3 = K_3;
log.K_4 = K_4;
log.K_5 = K_5;

%% Adaptive update
gamma_hat_dot = zeros(5,1);

if (norm(s)*norm(phi_1) < mu(1))
    gamma_hat_dot(1)=-varepsilon(1)*gamma_hat(1)+f(1)*norm(
        s)*norm(phi_1);
else
    gamma_hat_dot(1)=f(1)*norm(s)*norm(phi_1);
end

if (norm(s)*norm(phi_2) < mu(2))
    gamma_hat_dot(2)=-varepsilon(2)*gamma_hat(2)+f(2)*norm(
        s)*norm(phi_2);
else
    gamma_hat_dot(2) = f(2)*norm(s)*norm(phi_2);
end

```

```
80 if (norm(s)*norm(phi_3) < mu(3))
81     gamma_hat_dot(3)=-varepsilon(3)*gamma_hat(3)+f(3)*norm(
       s)*norm(phi_3);
82 else
83     gamma_hat_dot(3) = f(3)*norm(s)*norm(phi_3);
84 end
85
86 if (norm(s)*norm(phi_4) < mu(4))
87     gamma_hat_dot(4)=-varepsilon(4)*gamma_hat(4)+f(4)*norm(
       s)*norm(phi_4);
88 else
89     gamma_hat_dot(4) = f(4)*norm(s)*norm(phi_4);
90 end
91
92 if (norm(s)*norm(phi_5) < mu(5))
93     gamma_hat_dot(5)=-varepsilon(5)*gamma_hat(5)+f(5)*norm(
       s)*norm(phi_5);
94 else
95     gamma_hat_dot(5) = f(5)*norm(s)*norm(phi_5);
96 end
97
98 end
```

AdaptiveControlJoint.m

```
function [joint_torque, gamma_hat_dot_joint, log] = ...
    AdaptiveControlJoint(x,snake, theta_d, theta_dotd,
        theta_dotdotd,...
    gamma_hat_joint)

% Uses non-regressor based adaptive control algorithm to
    calculate the
% torque to put on the joint angles.

theta    = x(8 : 6+snake.n);
zeta     = x(7+snake.n : 11+2*snake.n);


%% Control constants and variables
sigma_joint = 10;
f = [10 10 10 3 3]';
k = 3*ones(length(snake.joint_vec),1);

theta_tilde = theta_d - theta;
theta_tilde_dot = theta_dotd - zeta(7:snake.n+5);
s = theta_tilde_dot + sigma_joint*theta_tilde;

epsilon = [.5 .25 .125 .0625 .03125]';
s =epsilon.*s;

mu = [0.4 0.4 2 0.2 0.2]';
varepsilon = [0.05 0.05 0.05 0.02 0.02]';
delta = 10;

phi_1 = theta_dotdotd;
phi_2 = zeta(7:snake.n+5);
phi_3 = k;
phi_4 = theta_tilde_dot;
phi_5 = theta_tilde;

K_1_joint = gamma_hat_joint(1)*s*phi_1'/max((norm(s)*norm(
    phi_1)),delta);
K_2_joint = gamma_hat_joint(2)*s*phi_2'/max((norm(s)*norm(
    phi_2)),delta);
K_3_joint = gamma_hat_joint(3)*s*phi_3'/max((norm(s)*norm(
    phi_3)),delta);
K_4_joint = gamma_hat_joint(4)*s*phi_4'/max((norm(s)*norm(
    phi_4)),delta);
K_5_joint = gamma_hat_joint(5)*s*phi_5'/max((norm(s)*norm(
```

```
    phi_5)),delta);

joint_torque = K_1_joint*phi_1 + K_2_joint*phi_2 +
    K_3_joint*phi_3...
     + K_4_joint*phi_4 + K_5_joint*phi_5;

%Sending values out to be plotted.
log.phi_1_joint = phi_1;
log.phi_2_joint = phi_2;
log.phi_3_joint = phi_3;
log.phi_4_joint = phi_4;
log.phi_5_joint = phi_5;

log.K_1_joint = K_1_joint;
log.K_2_joint = K_2_joint;
log.K_3_joint = K_3_joint;
log.K_4_joint = K_4_joint;
log.K_5_joint = K_5_joint;

% Saturating
sat = abs(joint_torque) > 50;
joint_torque(sat) = 50*sign(joint_torque(sat));

%Adaptive update
gamma_hat_dot_joint = zeros(5,1);

if (norm(s)*norm(phi_1) < mu(1))
    gamma_hat_dot_joint(1) = -varepsilon(1)*gamma_hat_joint
        (1) +...
         f(1)*norm(s)*norm(phi_1);
else
    gamma_hat_dot_joint(1) = f(1)*norm(s)*norm(phi_1);
end

if (norm(s)*norm(phi_2) < mu(2))
    gamma_hat_dot_joint(2) = -varepsilon(2)*gamma_hat_joint
        (2) +...
         f(2)*norm(s)*norm(phi_2);
else
    gamma_hat_dot_joint(2) = f(2)*norm(s)*norm(phi_2);
end

if (norm(s)*norm(phi_3) < mu(3))
    gamma_hat_dot_joint(3) = -varepsilon(3)*gamma_hat_joint
        (3) +...
```

```
79          f(3)*norm(s)*norm(phi_3);
80  else
81      gamma_hat_dot_joint(3) = f(3)*norm(s)*norm(phi_3);
82  end
83
84  if (norm(s)*norm(phi_4) < mu(4))
85      gamma_hat_dot_joint(4) = -varepsilon(4)*gamma_hat_joint
            (4) +...
86          f(4)*norm(s)*norm(phi_4);
87  else
88      gamma_hat_dot_joint(4) = f(4)*norm(s)*norm(phi_4);
89  end
90
91  if (norm(s)*norm(phi_5) < mu(5))
92      gamma_hat_dot_joint(5) = -varepsilon(5)*gamma_hat_joint
            (5) +...
93          f(5)*norm(s)*norm(phi_5);
94  else
95      gamma_hat_dot_joint(5) = f(5)*norm(s)*norm(phi_5);
96  end
97
98  end
```

getEulerAngles.m

```
function eulerAngles = getEulerAngles(R)
phi = atan2(R(3,2,1), R(3,3,1));
theta = -asin(R(3,1,1));
psi = atan2(R(2,1,1), R(1,1,1));

eulerAngles = [phi; theta; psi];
end
```

getTransformationMatrix.m

```
function T = getTransformationMatrix(eulerAngles)

phi = eulerAngles(1);
theta = eulerAngles(2);
psi = eulerAngles(3);

T = [1 sin(phi)*tan(theta) cos(phi)*tan(theta);...
     0 cos(phi) -sin(phi);...
     0 sin(phi)/cos(theta) cos(phi)/cos(theta)];
end
```

getEEpos.m

```
function [ee_state,ee_pos_b_ee] = getEEpos(snake,pos,theta,
    R)

ee_pos = [0; 0; 0;1];
ee_pos(1) = snake.link(snake.n).length;
%R_0ee = R(:,:,1);
for i=(snake.n-1):-1:1
    ee_pos = snake.joint(i).A(theta(i))*ee_pos;
    %R_0ee = R_0ee*R(:,:,i+1);
end
ee_pos = ee_pos(1:3);

%Position of end effector relative to body frame expressed
    in NED-frame
ee_pos_b_ee = R(:,:,1)*ee_pos;

%Position of end effector relative to inertial frame
    expressed in inertial
% frame
ee_pos = pos + R(:,:,1)*ee_pos;

R = R(:,:,snake.n);
%R = R_0ee;
phi_state = atan2(R(3,2), R(3,3));
theta_state = -asin(R(3,1));
psi_state = atan2(R(2,1), R(1,1));

ee_state = [ee_pos; phi_state; theta_state; psi_state];

end
```

plotResults.m

```
% This script uses the output from simulation to plot the interesting
% states, references, control output etc.

%% Taking the data out of struct format to simplify plotting
x_log = output.x_log;
tau_d_log = output.tau_d_log;
joint_torque_log = output.joint_torque_log;
x_ref_log = output.x_ref_log;
orientation_log = output.orientation_log;
T_log = output.T_log;
eta_ee_log = output.eta_ee_log;
eta_ee_ref_log = output.eta_ee_ref_log;
base_task = output.eta_base_task_log;
nominal_task = output.nominal_task_log;
theta_dotd_log = output.theta_dotd_log;
joint_controller_log = output.joint_controller_log;
phi_1_joint = joint_controller_log.phi_1;
phi_2_joint = joint_controller_log.phi_2;
phi_3_joint = joint_controller_log.phi_3;
phi_4_joint = joint_controller_log.phi_4;
phi_5_joint = joint_controller_log.phi_5;


%% Plot position of base link
figure(10)
subplot(3,1,1);
plot(T_log,x_log(1,:),'color','b');
hold on
plot(T_log,x_ref_log(1,:),'color','r');
hold on
plot(T_log,base_task(1,:),'color','c');
grid on
legend('\eta_1 (1)','\eta_{1,d} (1)', 'Task desired');
xlabel('t [s]');
ylabel('[m]');
title('X position of base link');

subplot(3,1,2);
plot(T_log,x_log(2,:),'color','b');
hold on
plot(T_log,x_ref_log(2,:),'color','r');
hold on
```

```
plot(T_log,base_task(2,:),'color','c');
grid on
legend('\eta_1 (2)','\eta_{1,d} (2)', 'Task desired');
xlabel('t [s]');
ylabel('[m]');
title('Y position of base link');

subplot(3,1,3);
plot(T_log,x_log(3,:),'color','b');
hold on
plot(T_log,x_ref_log(3,:),'color','r');
hold on
plot(T_log,base_task(3,:),'color','c');
grid on
legend('\eta_1 (3)','\eta_{1,d} (3)', 'Task desired');
xlabel('t [s]');
ylabel('[m]');
title('Z position of base link');

%% Plot orientation of base link
figure(20)
subplot(3,1,1);
plot(T_log,orientation_log(1,:),'color','b');
hold on
plot(T_log,x_ref_log(4,:),'color','r');
hold on
plot(T_log,base_task(4,:),'color','c');
grid on
legend('\phi','\phi_d', '\phi_{d,task}');
xlabel('t [s]');
ylabel('[rad]');
title('Roll angle of base link');

subplot(3,1,2);
plot(T_log,orientation_log(2,:),'color','b');
hold on
plot(T_log,x_ref_log(5,:),'color','r');
hold on
plot(T_log,base_task(5,:),'color','c');
grid on
legend('\theta','\theta_d', '\theta_{d,task}');
xlabel('t [s]');
ylabel('[rad]');
title('Pitch angle of base link');

```

```
88  subplot(3,1,3);
89  plot(T_log,orientation_log(3,:),'color','b');
90  hold on
91  plot(T_log,x_ref_log(6,:),'color','r');
92  hold on
93  plot(T_log,base_task(6,:),'color','c');
94  grid on
95  legend('\psi','\psi_d', '\psi_{d,task}');
96  xlabel('t [s]');
97  ylabel('[rad]');
98  title('Yaw angle of base link');
99
100 %% Plot position of end effector
101 figure(50)
102 subplot(3,1,1);
103 plot(T_log,eta_ee_log(1,:),'color','b');
104 hold on
105 plot(T_log,eta_ee_ref_log(1,:),'color','c');
106 grid on
107 legend('x_{ee}', 'x_{ee,d,task}');
108 xlabel('t [s]');
109 ylabel('[m]');
110 title('X position of end effector');
111
112 subplot(3,1,2);
113 plot(T_log,eta_ee_log(2,:),'color','b');
114 hold on
115 plot(T_log,eta_ee_ref_log(2,:),'color','c');
116 grid on
117 legend('y_{ee}', 'y_{ee,d,task}');
118 xlabel('t [s]');
119 ylabel('[m]');
120 title('Y position of end effector');
121
122 subplot(3,1,3);
123 plot(T_log,eta_ee_log(3,:),'color','b');
124 hold on
125 plot(T_log,eta_ee_ref_log(3,:),'color','c');
126 grid on
127 legend('z_{ee}', 'z_{ee,d,task}');
128 xlabel('t [s]');
129 ylabel('[m]');
130 title('Z position of end effector');
131
132 %% Plot orientation of end effector
```

```
figure(60)
subplot(3,1,1);
plot(T_log,eta_ee_log(4,:),'color','b');
hold on
plot(T_log,eta_ee_ref_log(4,:),'color','c');
grid on
legend('\phi_{ee}', '\phi_{ee,d,task}');
xlabel('t [s]');
ylabel('[rad]');
title('Roll angle of end effector');

subplot(3,1,2);
plot(T_log,eta_ee_log(5,:),'color','b');
hold on
plot(T_log,eta_ee_ref_log(5,:),'color','c');
grid on
legend('\theta_{ee}', '\theta_{ee,d,task}');
xlabel('t [s]');
ylabel('[rad]');
title('Pitch angle of end effector');

subplot(3,1,3);
plot(T_log,eta_ee_log(6,:),'color','b');
hold on
plot(T_log,eta_ee_ref_log(6,:),'color','c');
grid on
legend('\psi_{ee}', '\psi_{ee,d,task}');
xlabel('t [s]');
ylabel('[rad]');
title('Yaw angle of end effector');

%% Plot joint angles
%Plot joint angles

figure(100)
n_joint = snake.n-1;
for i=1 : n_joint
    %figure(100+i)
    subplot(n_joint,1,i);
    plot(T_log,x_log(7+i,:),'color','b');
    hold on
    plot(T_log,x_ref_log(6+i,:),'color','r');
%     hold on
%     plot(T_log,nominal_task(i,:),'color','c');
    grid on
```

```
178     legend(['\theta_{' num2str(i) '}'],['\theta_{' num2str(
            i)...
179         ',d}']);%, ['\theta_{' num2str(i) ',task}']);
180     xlabel('t [s]');
181     ylabel('[rad]');
182     title(['Angle of joint ' num2str(i)]);
183 end
184
185 %% Plot translational forces on base link from control law
186 figure(210)
187 subplot(3,1,1);
188 plot(T_log,tau_d_log(1,:),'color','b');
189 grid on
190 legend('\tau (1)');
191 xlabel('t [s]');
192 ylabel('[N]');
193 title('Thrust in surge from base link controller');
194
195 subplot(3,1,2);
196 plot(T_log,tau_d_log(2,:),'color','b');
197 grid on
198 legend('\tau (2)');
199 xlabel('t [s]');
200 ylabel('[N]');
201 title('Thrust in sway from base link controller');
202
203 subplot(3,1,3);
204 plot(T_log,tau_d_log(3,:),'color','b');
205 grid on
206 legend('\tau (3)');
207 xlabel('t [s]');
208 ylabel('[N]');
209 title('Thrust in heave from base link controller');
210
211 %% Plot moments applied by the control law
212 figure(211)
213 subplot(3,1,1);
214 plot(T_log,tau_d_log(4,:),'color','b');
215 grid on
216 legend('\tau (4)');
217 xlabel('t [s]');
218 ylabel('[Nm]');
219 title('Moment in roll from base link controller');
220
221 subplot(3,1,2);
```

```
222 plot(T_log,tau_d_log(5,:),'color','b');
223 grid on
224 legend('\tau (5)');
225 xlabel('t [s]');
226 ylabel('[Nm]');
227 title('Moment in pitch from base link controller');
228
229 subplot(3,1,3);
230 plot(T_log,tau_d_log(6,:),'color','b');
231 grid on
232 legend('\tau (6)');
233 xlabel('t [s]');
234 ylabel('[Nm]');
235 title('Moment in yaw from base link controller');
236
237 %% Plot tau for forces on joints
238 %Forces on joints
239 figure(400)
240 for i=1 : snake.n-1
241     subplot(snake.n-1,1,i);
242     plot(T_log,joint_torque_log(i,:),'color','b');
243     grid on
244     legend(['\tau_{joint} (' num2str(i) ')']);
245     xlabel('t [s]');
246     ylabel('[Nm]');
247     title(['Torque on joint ' num2str(i)]);
248 end
249
250
251 %% Plot joint velocities
252 figure(600)
253 for i = 1:snake.n-1
254     subplot(snake.n-1,1,i);
255     plot(T_log,x_log(12+snake.n+i,:),'color','b');
256     hold on
257     plot(T_log,x_ref_log(11+snake.n+i,:),'color','m');
258     grid on
259     legend(['Velocitiy of joint  ' num2str(i)],'Desired
        velocity');
260     xlabel('t [s]');
261     ylabel('[rad/s]');
262     title(['Velocity of joint  ' num2str(i)]);
263 end
```

Jonas Hopsdal

Adaptive Control of Underwater Snake Robot

NTNU
Norwegian University of
Science and Technology