NTNU

NORWEGIAN UNIVERSITY OF SCIENCE AND TECHNOLOGY

DEPARTMENT OF ENGINEERING CYBERNETICS

Specialization Project Report

Simulator for Obstacle Aided Locomotion in Snake Robots

Author:	Atussa Koushan
Supervisor:	Øyvind Stavdahl
Submission Date:	December 17, 2019

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

Prosjektoppgave

Studentens navn:	Atussa Koushan
Fag:	Teknisk kybernetikk
Tittel (norsk):	Simulaor for hindringsbasert fremdrift hos slangeroboter
Tittel (English):	Simulator for obstacle aided locomotion in snake robots

Description:

A number of different principles for snake robot locomotion have been proposed and tested, many of which are based on heuristic rules and stiff position controlled joints. A more physics-based and compliant method is being developed which is based on the formalism of *hybrid position/force control*. In this assignment you will study this emerging technique and simulate an idealized two-dimensional snake robot propelled by contact forces based on hybrid position/force control in an idealized environment comprising frictionless movement and extentionless environmental assets ("obstacles").

1. Study the theoretical backdrop of the assignment, and provide a brief overview of a) hybrid position-/force control (HPFC) in general, and b) HPFC applied to snake robots in particular. Special emphasis should be put on possibilities and limitations associated with HPFC in snake robot applications.
2. Establish a suitable model for the idealized robot and the robot's environment. Describe and justify any assumptions and simplifications made.
3. Develop a simulation platform for OAL based on HPFC concepts. To the extent possible, demonstrate relevant simulation scenarios.

Veileder(e): Øyvind Stavdahl, Institutt for teknisk kybernetikk

Trondheim, 05.09.2019

Øyvind Stavdahl
Faglærer

Abstract

Snake robots are robust serial link robots able to propel themselves through uneven and irregular terrain. As opposed to traditional mobile robots, the snake robot can utilize obstacles found in the environment to push itself forward along a predefined path. The aforementioned mode of locomotion is referred to as **obstacle aided locomotion (OAL)**. The challenge of both controlling the shape and contact forces of the snake robot has motivated the study of **hybrid position/force control (HPFC)**, with special emphasis on application within OAL.

The main principle of HPFC together with a simplified 2-dimensional model of the snake robot and its environment are the backbone of the developed MATLAB simulator. The objective of this simulator is to visualize the behavior of the snake robot and OAL in simple path following scenarios.

Relevant experiments conducted during this study has proven the simulator to be a great resource for presenting concepts and study limitations and possibilities within OAL. However, a geometrical approximation was made to the applied part of the HPFC concept during development of the simulator, which has led to some deviations from the true dynamics of the system and the laws of conservation of energy and momentum to be violated. This is further discussed in the report.

The code of the simulation program can be provided upon request.

Sammendrag

Slangeroboter er robuste seriekoblede roboter som evner å traversere ujevne og uregelmessige terreng. I motsetning til tradisjonelle mobile roboter, kan slangeroboter utnytte hindringer i miljøet sitt ved å dytte seg selv fram langs en forhåndsdefinert bane. Denne bevegelsesarten blir omtalt som **obstacle aided locomotion (OAL)** eller hindringsbasert fremdrift. Utfordringen som trer fram ved kontrollering av både formen og kontaktkreftene til slangeroboten har motivert studiet av **hybrid position/force control (HPFC)** eller hybrid posisjons- og kraftstyring, med spesiell vekt på applikasjoner innenfor OAL.

Hovedprinsippet bak HPFC kombinert med en todimensjonal modell av slangen og dens miljø er selve fundamentet til den utviklede MATLAB-simulatoren. Formålet med denne simulatoren er å visualisere slangerobotens oppførsel og OAL i enkle banefølgingsscenarioer.

Simulatoren har fra relevante eksperimenter utført under studiet vist seg å være et godt verktøy for presentering av konsepter og studere både begrensninger og muligheter knyttet til OAL. På en annen side har en geometrisk approksimasjon av den anvendte delen av HPFC i simulatoren ført til noen avvik fra den sanne dynamikken i systemet og brudd med lovene om konser-

vasjon av energi og bevegelsesmenge. Dette er ytterligere diskutert i rapporten.

Koden for simuleringsprogrammet er til disposisjon ved ønske.

Preface

This report covers the work of the specialization project for the masters study Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). The work has been conducted with great counselling from supervisor Øyvind Stavdahl. A lot of Stavdahl's previous work and notes also make out the mathematical foundation of the project. Furthermore, Irja Gravdahl has contributed to valuable discussions on the topic.

All simulator code is written from scratch by me. The figures presented throughout the report are also designed by me, unless stated otherwise. When it comes to data presentation, Åsmund Eek has provided great advice on generation of plots in MATLAB.

Atussa Koushan
17.12.2019
Trondheim, Norway

Contents

Abstract v

Sammendrag vii

Preface ix

Nomenclature xv

List of Tables xix

List of Figures xxi

1 Introduction 1

1.1 Previous work 1

1.2 Scope of the project 2

1.2.1 Simplifications 2

1.2.2 Contributions 3

1.3 Report structure 4

2 Model specifications **5**
2.1 Assumptions 5
2.2 Further model description 6
    2.2.1 The snake robot 6
    2.2.2 The environment 7
    2.2.3 The obstacles 7
    2.2.4 The optimal path 8

3 Background theory **9**
3.1 Snake robot kinematics 9
    3.1.1 Constrained kinematics 13
3.2 Snake robot dynamics 15
    3.2.1 Constrained dynamics 18
    3.2.2 Momentum 19
3.3 Computed torque control 20
    3.3.1 PD control 20
    3.3.2 Feedforward control 21
    3.3.3 Feedforward and feedback linearization 21
3.4 Obstacle aided locomotion (OAL) 22
3.5 Hybrid position/force control (HPFC) 23
    3.5.1 Constrained motion 24
    3.5.2 Multiple constraints 26
    3.5.3 Passive joints 27
    3.5.4 Dynamic HPFC 27
    3.5.5 HPFC for snake robots 29
3.6 Projection onto path 31

3.7 Contact detection 33

4 The simulator 35

4.1 Simplifications and limitations 36
4.2 Program flow 37
4.3 User guide 37
4.4 Program code summary 41
4.4.1 Dynamics functions 41
4.4.2 Kinematics functions 41
4.4.3 Contact Jacobian function 41
4.4.4 Projection function 43
4.4.5 Control 44
4.4.6 Visualization 45

5 Experiments 47

5.1 Obstacle-free environment 48
5.1.1 Single reference computed torque control 48
5.1.2 Path alignment 50
5.2 Collision analysis 54
5.2.1 Single obstacle interaction 54
5.3 OAL simulation scenarios 57
5.3.1 Simple propulsion 57
5.3.2 Obstacle interaction without propulsion 61
5.3.3 Unsuccessful propulsion attempt along path 62
5.3.4 Propulsion along path 64

6 Discussion **69**
6.1 Effects of the contact behaviour simplification 69
6.2 Review of the path alignment method 72
6.3 Further insights from experiments 73

7 Conclusion **75**
7.1 Future work 76

Bibliography **77**

Nomenclature

The following list describes several symbols and abbreviations used in the report. All units follow the SI unit system.

Abbreviations

HPFC	Hybrid Position/Force Control
OAL	Obstacle Aided Locomotion
HOAL	Hybrid Obstacle Aided Locomotion

Control symbols

K_p	Proportional gain
K_d	Derivative gain
ζ	Damping ratio
ω_n	Natural frequency
$\mathbf{q_d}$	Desired joint angles
$\mathbf{q_e}$	Joint angle errors

Robot dynamics symbols

$\boldsymbol{\tau}$	Generalized torques

$\boldsymbol{\tau}_m$	Joint motor torques
$\boldsymbol{\tau}_c$	Joint torques from external forces
$\mathbf{M}$	Mass matrix
$\mathbf{C}$	Coriolis matrix
$\mathbf{g}$	Gravity matrix
L	Lagrangian
K, P	Kinetic and potential energy
I	Moment of inertia of rod
$\mathbf{f}^b_{c,i}, \mathbf{f}^i_{c,i}$	External force on link i in base frame and frame of link i
α_i	Angle of link i relative to base frame
$\mathbf{r}$	End effector position in the base frame b
$p(\mathbf{r})$	Constraint hypersurface
$\mathbf{E_F}$	Matrix of hypersurface unit normal vectors
$\mathbf{f}^b$	End effector force on hypersurface
$\mathbf{p}$	Momentum

Robot kinematics symbols

n	Number of links
N	Number of generalized coordinates
m	Link mass
l	Link length
$\mathbf{q}$	Generalized coordinates

ϕ_i	Joint angle of link i
(x_0, y_0)	Position of tail in base frame
(x_i, y_i)	Position of endpoint of link i in base frame b
$(x_{c,i}, y_{c,i})$	Position of contact point on link i in base frame b
$d_{c,i}$	Distance from joint i to contact point on link i
$\mathbf{T_{bi}}$	Transformation matrix from base frame to frame of link i
$\mathbf{T_{bci}}$	Transformation matrix from base frame to frame of contact point on link i
$\mathbf{v_i}$	Velocity of endpoint of link i in base frame b
$\mathbf{v_{c,i}}$	Velocity of contact point on link i in base frame b
$\mathbf{J}$	Jacobian matrix
$\mathbf{J_c}$	Jacobian matrix for contact point
θ	Joint angle deviation from path
r_{safety}	Radius around obstacle

Spaces

$\mathbb{T}$	Task space
$\mathbb{F}$	Force space
$\mathbb{M}$	Motion space
$\mathbb{C}$	Constraint space
$\mathbb{P}$	Propulsion space
$\mathbb{S}$	Shape space

List of Tables

4.1 User adjustable simulator parameters 40

5.1 Common simulation configuration for all cases 47
5.2 Simulation configuration for 5.1.1 48
5.3 Simulation configuration for 5.1.2 50
5.4 Simulation configuration for 5.2.1 54
5.5 Simulation configuration for 5.3.1 58
5.6 Simulation configuration for 5.3.2 61
5.7 Simulation configuration for 5.3.3 63
5.8 Simulation configuration for 5.3.4 65

List of Figures

2.1 Model of snake robot with n links 7

2.2 Model of snake robot and obstacles 8

3.1 Model of snake robot with notation 10

3.2 Snake robot in contact with obstacle 14

3.3 Computed torque control block diagram 22

3.4 Configuration of the snake robot and obstacles yielding no propulsion 30

3.5 Shortest distance from joint to path [16] 32

3.6 Robot link and obstacle with safety radius 34

4.1 Program flow diagram 38

4.2 Visual output of simulation 39

5.1 Simulation demo - computed torque control 49

5.2 Simulation demo - adjusting to path from nearby 51

5.3 Simulation demo - adjusting to path from a distance 52

5.4 Joint angles for path adjustment with different starting configurations . . . 53
5.5 Energy, momentum and joint motor torques of robot for single collision scenario . . . 55
5.6 Simulation demo - interaction with single obstacle . . . 56
5.7 Simulation demo - propulsion with static joint setpoint . . . 59
5.8 Joint angles and velocities for the single setpoint scenario . . . 60
5.9 Simulation demo - no propulsion . . . 62
5.10 Simulation demo - failed propulsion along path . . . 64
5.11 Simulation demo - propulsion along path . . . 67

Chapter 1

Introduction

1.1 Previous work

The department of engineering cybernetics at the Norwegian University of Science and Technology (NTNU) has made significant contributions to the field of snake robot control, related to both aquatic snake-like propulsion and efficient snake robot locomotion on flat surfaces [1].

However, snakes can also utilize irregularities in a terrain. Consequently, Transeth et al. [2] suggested the term *Obstacle Aided Locomotion (OAL)* for snake robots that actively use walls or external objects for means of propulsion.

A relevant concept for attainment of this type of propulsion is *Hybrid Position/Force Control (HPFC)*. The concept was first introduced by Raibert and Craig et al. in 1981 [3]. West and Asada [4] further proposed a method for the design of HPFC controllers in 1985 for constrained manipulators that are in contact with the environment at multiple points. This method aims at controlling the position and force at the manipulator joints with kinematics-based projections such that the motion at the end effector, and the force at the contacts with the environment are those required for performing the task. Yoshikawa [5] advanced the constraint analysis by taking the dynamics into consideration.

Stavdahl [1] proposed the combination of OAL and HPFC, leading to the term *Hybrid Obstacle Aided Locomotion (HOAL)*. It is still a small area of research, but has a lot of potential in the area of e.g. rescue robots in cluttered environments. Klafstad [6] later summarized the concept of HPFC in snake robots with special emphasis on the method of West and Asada [4].

1.2 Scope of the project

The goal of this project is to establish a suitable and simplified 2-dimensional model of the idealized robot and its environment. A simulator is then to be developed from scratch based on this model and the concepts of OAL and HPFC. The purpose of the simulator is from the project task interpreted to be a platform for showing the aforementioned concepts on snake robots, rather than data generation for physical purposes. Seeing as HOAL still is a fresh area of research and has little solid theoretical backdrop, this work goes in the direction of a proof-of-concept and experiments were performed to verify the method.

1.2.1 Simplifications

Due to the strict time constraints of the project and somewhat vague outlines, simplifications have been made to provide for a specific end goal. The simulator is restricted to handle bounded and well-defined scenarios with slow dynamics. Furthermore, the calculation of interaction forces in the simulator is neglected in favor of an easier discrete projection method that only takes joint velocities into consideration. This geometrical approximation is explained in further detail in Chapter 4.

The simplifications have led to some deviations from the true dynamical behavior of the robot when it is in contact with obstacles. As the velocity is changed without regard to energy conservation, this is most prominent in cases where the velocity is increased rather than decreased. An example is given from the experiment in 5.2.1, where the energy is close to tripled after contact with a single obstacle. This is however not a very dominating factor in the experiments because the project, as mentioned, is focused on slow dynamics of the model. The consequences of the approximation are discussed further in Chapter 6.

All assumptions regarding the employed model can be found in Chapter 2.

1.2.2 Contributions

A lot of the contributions connected to the limited research environment of HOAL have been clarification of the associated formulations and theoretical aspects. Additionally, specific contributions are

- Study and analysis of HPFC, both with and without a dynamics consideration.
- Develpoment of a mathematical model of the snake robot and its environment related to the given problem formulation.

- MATLAB simulator for demonstration of simple and germane HOAL-scenarios. It should be noted that this simulator is programmed from scratch and made in the context of an experimental study. On the basis of the generalized structure of the simulator program, several of the related modules can be applied to future implementations.
- Testing of the simulator performance and an evaluation of the associated limitations.
- Evaluation of the OAL concept and discussion of remaining challenges, as well as proposals for improvement.

1.3 Report structure

It is assumed that the reader is familiar with basic robotics and control theory. Please see [7], [8], [9], [10] for a more thorough explanation of the topics.

The report first introduces the specifics around the mathematical model used for the snake robot, its environment and task in Chapter 2. Further, a review of the required background theory for understanding the employed dynamical model and control of the snake robot is presented in Chapter 3. Chapter 4, is focused on the simulator and method used for achieving OAL. Significant parts of the simulator are explained in detail and a guide for usage is provided. Chapter 5 presents some experiments performed with the simulator, which are discussed in Chapter 6 with focus on limitations and possibilities related to both the simulator and OAL in general. Lastly, Chapter 7 concludes the work and briefly proposes some ideas for future improvements based on the challenges encountered in this project.

Chapter 2

Model specifications

This chapter presents the physical aspects of the model used throughout the report. The model has four main parts, namely the environment, the snake robot, the obstacles and the desired path. The goal of the snake robot is at all times to interact with the obstacles close to the path and utilize them to propel itself forward along the path. The value of all variables needed to express the model and problem are assumed known at all times.

2.1 Assumptions

Assumptions 1-9 are taken from [1], whilst assumptions 10-11 are specific for this project.

1. All parts of the model are assumed to be rigid
2. The robot has n joints and all links have length l and mass m.

3. Only flat, 2-dimensional cases are considered.
4. The robot has no lateral extension.
5. There is no friction.
6. Obstacles have no spatial extent, only a static position in the plane.
7. There exists a predetermined desired continuous path S.
8. The robot is considered to be "on the path" whenever all joint centers coincide with the path.
9. The robot is initially on or close to the path.
10. Any link is in contact with at most one obstacle at the time.
11. All energy from a link colliding with another link is dissipated.

2.2 Further model description

2.2.1 The snake robot

The snake robot itself is modeled as a simple planar robot manipulator with links and joints. The main difference between the snake robot and a classic robot manipulator is the property that the snake robot is not physically attached to any fixed point in the world. This frees one constraint. However, it is still relevant to express the position of the last link of the snake, also denoted as the tail. This is performed by introducing three virtual joints to the model; two translational and one rotational joint. These joints are not controllable and merely for describing the kinematics, dynamics and constraints. The model of

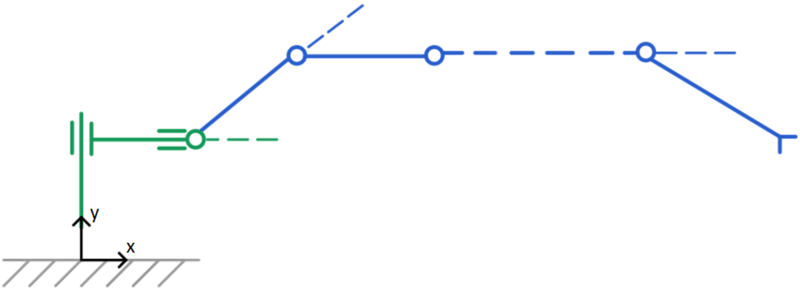


Figure 2.1: Model of snake robot with n links

the snake robot is visualized in Figure 2.1, where real robot links are blue and the virtual ones are green.

2.2.2 The environment

The environment is the (x,y)-plane in Figure 2.1, and consists of nothing but the robot and the obstacles.

2.2.3 The obstacles

The obstacles are modeled as rigid points in the plane. However, as a consequence of the discrete nature of the simulator, there is a "safety barrier" around every obstacle defining the area in which contact is present. This contact is still modeled as a point contact. The barriers can be seen as the red circles in Figure 2.2. The points the circles are surrounding are the actual obstacles.

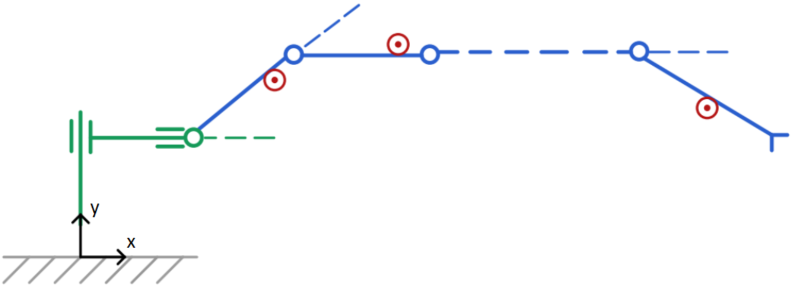


Figure 2.2: Model of snake robot and obstacles

2.2.4 The optimal path

The path is in this report made out of straight lines and quadrants with radius equal to two times the link length l. There is no gap between these segments, following from the assumption of a continuous path.

Chapter 3

Background theory

This chapter introduces relevant kinematics, dynamics and control for the snake robot specific case used in the development of the simulator. Further, the concepts of OAL and HPFC are explained to give insight into the motivation and structure of the simulator. Even though the idea of dynamical HPFC [5] is not applied in the project, it is presented as it is an essential part of the discussions in Chapter 6. Lastly, the methods for detecting contact with obstacles and adjusting to a predefined path are explained.

3.1 Snake robot kinematics

The snake robot is modeled as a serial chain, which is a system of rigid bodies in which each member is connected to two others, except for the first and last members that are each connected to only one other member [9]. As opposed to traditional robot manipulator models, the first joint in the snake robot model

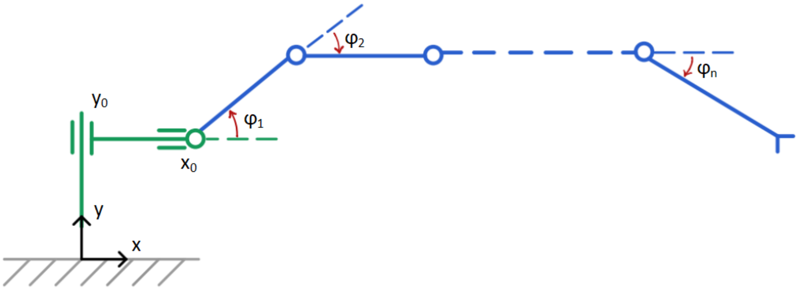


Figure 3.1: Model of snake robot with notation

is not physically connected to a base.

The vector of generalized coordinates $\mathbf{q}$ for a snake robot with n links is

$$\mathbf{q} = \begin{bmatrix} \phi_1 & \phi_2 & \dots & \phi_n & x_0 & y_0 \end{bmatrix}^T . \tag{3.1}$$

The coordinates (x_0, y_0) and ϕ_1 represent the position and orientation of the tail of the snake robot in reference to the base frame (x, y). These coordinates cannot be directly controlled and will therefore be referred to as virtual coordinates. The generalized coordinates $\phi_2, \dots, \phi_n$, corresponding to the actuated joints, refer to the angle of the following link relative to the preceding link. The number of generalized coordinates including two position coordinates and n joint angles is $N = n + 2$.

The model of the snake robot with the named variables are illustrated in Figure 3.1.

Homogeneous transformation matrices are used to express the pose (position and orientation) of the links in relation to the base frame. This means that as long as the joint angles and size of the snake robot are known, the Cartesian positions can be calculated. The homogeneous transformation matrix for the end point of link i from the base frame b is given by (3.2). The base frame will stay put regardless of motion of the robot. Note also that the link length l is assumed equal for all the links.

$$\mathbf{T}_{bi} = \mathbf{D}_x(x_0)\mathbf{D}_y(y_0)\sum_{k=1}^{i}\mathbf{R}_z(\phi_k)\mathbf{D}_x(l) \tag{3.2}$$

The translation and rotation matrices are given by

$$\begin{gathered}\mathbf{D}_x(x) = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{D}_y(y) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}, \\ \mathbf{R}_z(\phi) = \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \end{gathered} \tag{3.3}$$

The transformation matrix from the reference frame to the center of link i can be found in the same manner. The only difference is that the very last translational matrix has to take the argument $l/2$ instead of l. This is useful to keep in mind as it be used in Section 3.2 for the derivation of the kinetic energy of the links.

As mentioned earlier, the transformation matrix $\mathbf{T}_{bi}$ can be used to find the absolute orientation and position of the tip of link i in the base frame. The resulting matrix can be written on the form

$$\mathbf{T}_{bi} = \begin{bmatrix} \mathbf{R}_{bi}(\phi_{i,abs}) & \mathbf{t}^r_{ri} \\ \mathbf{0}^T & 1 \end{bmatrix}. \tag{3.4}$$

The position is directly extracted from $\mathbf{t}^r_{ri} = [x_i y_i]^T$. The orientation $\phi_{i,abs}$ is found by comparing $\mathbf{R}_{bi}$ to $\mathbf{R}_z$ and solving for $\phi_{i,abs}$.

Alternatively, one can directly compute the position of the center of a link i from the expressions below

$$\begin{aligned} x_i &= x_0 + \sum_{k=1}^{i} l \cos\,(\sum_{j=1}^{k} \phi_j) \\ y_i &= y_0 + \sum_{k=1}^{i} l \sin\,(\sum_{j=1}^{k} \phi_j), \end{aligned} \tag{3.5}$$

where l is the link length and $1 \leq i \leq n$.

Forward and inverse instantaneous kinematics

The well known Jacobian lets us transform between Cartesian and joint velocities. It is derived by taking the partial derivative of the x and y position of link $1 \leq i \leq n$ with respect to all generalized coordinates

$$\mathbf{J_i} = \begin{bmatrix} \frac{\partial x_i}{\partial q_1} & \cdots & \frac{\partial x_i}{\partial q_{N-1}} & \frac{\partial x_i}{\partial q_N} \\ \frac{\partial y_i}{\partial q_1} & \cdots & \frac{\partial y_i}{\partial q_{N-1}} & \frac{\partial y_i}{\partial q_N} \end{bmatrix}. \tag{3.6}$$

The relationship between the Cartesian velocity $\mathbf{v}$ of the point (x_i, y_i) on the robot and the joint velocities $\dot{\mathbf{q}}$ can thus be written as

$$\mathbf{v_i} = \mathbf{J_i}(\mathbf{q})\dot{\mathbf{q}} \quad \text{and} \quad \dot{\mathbf{q}} = \mathbf{J_i}(\mathbf{q})^{\dagger}\mathbf{v_i}. \tag{3.7}$$

The first equation is formally referred to as the forward instantaneous kinematics, whereas the second one is referred to as the inverse instantaneous kinematics. $\mathbf{J}(\mathbf{q})^{\dagger}$ is the pseudo inverse of the Jacobian, which has to be used as a result of the Jacobian being non-square.

3.1.1 Constrained kinematics

For the case in which the robot is in contact with the environment, the motion will be constrained. The obstacles found in the environment are modelled as single frictionless points. The only constraint imposed by the environment is that the robot cannot penetrate the obstacles. It can, however, both apply an arbitrary large force against them or move along them.

The model assumes that any link can be in contact with at most one obstacle at the time. To represent the mentioned constraint, the vector of generalized coordinates is expanded with n further elements, where n is the number of links. The updated vector $\mathbf{q}$ is now

$$\mathbf{q} = \begin{bmatrix} \phi_1 & \phi_2 & ... & \phi_n & x_0 & y_0 & d_{c,1} & d_{c,2} & ... & d_{c,n} \end{bmatrix}^T, \tag{3.8}$$

where $N = 2 + 2n$ is the new number of generalized coordinates.

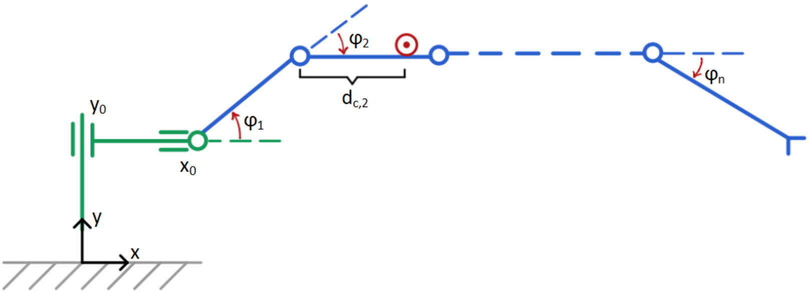


Figure 3.2: Snake robot in contact with obstacle

The newly introduced coordinates $d_{c,1}, ..., d_{c,n} \geq 0$ represent the distance to the possible contact point from the corresponding joint. For instance, the coordinate $d_{c,2}$ is the distance between the second joint and the contact point measured along the second link, as illustrated in Figure 3.2. Every link might not be in contact with an obstacle, but the maximum number of coordinates is introduced in the interest of keeping the vector size constant. Seeing as there is no actuation force directly connected to the obstacle-related coordinates, they will be referred to as virtual joints or virtual coordinates.

The position of a contact point on link $1 \leq i \leq n$ in the base frame can be derived through the corresponding transformation matrix (3.9).

$$\mathbf{T}_{bci} = \mathbf{D}_x(x_0)\mathbf{D}_y(y_0)\sum_{k=1}^{i-1}(\mathbf{R}_z(\phi_k)\mathbf{D}_x(l))\mathbf{R}_z(\phi_i)\mathbf{D}_x(d_{c,1}) \tag{3.9}$$

Constrained instantaneous kinematics

The Jacobian matrix related to the velocity of the contact point can be derived in the same manner as in the unconstrained case. The only difference is that the partial differentiation of the contact point (x_c, y_c) is now taken with respect to the extended vector of generalized coordinates (3.8). The resulting contact Jacobian for a contact point on link $1 \leq i \leq n$ is thus

$$J_{c,i} = \begin{bmatrix} \frac{\partial x_{c,i}}{\partial \phi_2} & \cdots & \frac{\partial x_{c,i}}{\partial q_{N-1}} & \frac{\partial x_{c,i}}{\partial q_N} \\ \frac{\partial y_{c,i}}{\partial \phi_2} & \cdots & \frac{\partial y_{c,i}}{\partial q_{N-1}} & \frac{\partial y_{c,i}}{\partial q_N} \end{bmatrix}. \tag{3.10}$$

This Jacobian will end up being quite sparse, seeing as the coordinate of a contact point is independent of all other contact coordinates. This is a property that can be exploited using sparse solvers if the snake robot has a large number of links.

The relationships between the Cartesian velocity of a contact point on link $1 \leq i \leq n$ and the joint velocities can now be expressed as

$$\mathbf{v_{c,i}} = \mathbf{J_{c,i}}(\mathbf{q})\dot{\mathbf{q}} \quad \text{and} \quad \dot{\mathbf{q}} = \mathbf{J_{c,i}}(\mathbf{q})^{\dagger}\mathbf{v_{c,i}}. \tag{3.11}$$

3.2 Snake robot dynamics

The snake robot has $n - 1$ joint actuators that all can apply torques to their corresponding joints. The dynamics describe how the robot moves in response to these actuator forces. For simplicity, it is assumed that the actuators do not have dynamics of their own and, hence, arbitrary torques can be commanded at the joints of the robot [11].

The dynamics of the snake robot will be expressed using the joint space equations of motion formulation

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}. \tag{3.12}$$

Because the movement is restricted to the 2D plane, the gravitational term $\mathbf{g}(\mathbf{q})$ can be neglected and the equations of motion can be rewritten to

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\tau}, \tag{3.13}$$

where $\mathbf{M}(\mathbf{q})$ and $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the mass matrix and Coriolis matrix respectively. $\boldsymbol{\tau}$ is the vector of generalized torques corresponding to the generalized coordinates (3.1). Furthermore, the elements corresponding to the virtual coordinates will be zero at all times.

Solving (3.13) with respect to $\ddot{\mathbf{q}}$ yields

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})(\boldsymbol{\tau} - \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})). \tag{3.14}$$

Several methods exist for finding the equations of motion for a robot. The Euler-Lagrange method [7], which is chosen here, is based on the difference in kinetic energy (K) and potential energy (P) of the system, also known as the Lagrangian

$$L = K - P. \tag{3.15}$$

The equations of motion can now be expressed as a second order partial differential equation

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{\mathbf{q}}} - \frac{\partial L}{\partial \mathbf{q}} = \boldsymbol{\tau}. \tag{3.16}$$

Again, simplifications can be made from the restricted movement in the world and thus the potential energy P can be neglected. The Lagrangian is therefore simply equal to the kinetic energy, which is the sum of the kinetic energy for every link [12]. Furthermore, the kinetic energy for one link i is divided into two parts, $K_{translational}$ and $K_{rotational}$. The kinetic energy can now be express as

$$K = \sum_{i=1}^{n} (K_{translational,i} + K_{rotational,i}), \tag{3.17}$$

where the translational and rotational kinetic energy is given in (3.18) and (3.19) respectively.

$$K_{translational,i} = \frac{1}{2} m (\dot{x}_i^2 + \dot{y}_i^2) \tag{3.18}$$

Here m is the link mass, and $(\dot{x}_i, \dot{y}_i)$ make out the velocity of the center of the link found by differentiating (3.5) with respect to time.

$$K_{rotational,i} = \frac{1}{2} I \dot{\phi_i}^2 \tag{3.19}$$

$\dot{\phi}_i$ is the joint velocity of link i. Furthermore, every link has the same moment of inertia, namely $I = (1/12)ml^2$. This is the moment of inertia of a rod, corresponding to the moment of inertia of a cylinder with zero radius [7].

3.2.1 Constrained dynamics

The generalized torques from the right side of (3.13) can be split into two parts whenever there is contact between the robot and the environment, namely a component resulting from the control inputs (motor torques), $\boldsymbol{\tau}_m$, and a component resulting from the external forces acting on the robot, $\boldsymbol{\tau}_c$ [12]. The generalized torques can thus be written

$$\boldsymbol{\tau} = \boldsymbol{\tau}_m + \boldsymbol{\tau}_c. \tag{3.20}$$

According to Holden et al. [13], the force from an obstacle acting on a link is two-fold: one normal to the link and one tangent to the link. The force tangent to the link is due to friction and will therefore be neglected in this project. The remaining normal force is preventing the link from moving into the obstacle when the robot itself is applying a force to the obstacle. The external force in the frame of link i acting on link i is denoted $\mathbf{f}^i_{c,i}$ and can be written as

$$\mathbf{f}^i_{c,i} = \begin{bmatrix} 0 \\ f^i_{c,i,y} \end{bmatrix}. \tag{3.21}$$

The following derivations are inspired by [12]. The force $\mathbf{f}^i_c$ can be expressed in the base frame by using the rotation matrix from (3.3):

$$\mathbf{f}^b_{c,i} = \mathbf{R}_z(\alpha_i)\mathbf{f}^i_{c,i}, \tag{3.22}$$

where α_i is the angle of link i related to the base frame b. It can be found by

$$\alpha_i = \sum_{k=1}^{i} \phi_k. \tag{3.23}$$

The contact Jacobian (3.10) can be used to find the generalized external torque as a result of the external forces:

$$\boldsymbol{\tau_c} = \sum_{i=1}^{n} \mathbf{J}_{c,i}^{T} \mathbf{f}_{c,i}^{b}. \tag{3.24}$$

3.2.2 Momentum

The calculation of the momentum of the robot is presented as it is used in the experiment in 5.2.1 in validation of the mathematical model employed in the project.

Momentum is the product of the velocity and mass of an object

$$\mathbf{p} = m\mathbf{v}. \tag{3.25}$$

For a snake robot with multiple links, which can be seen as a system of objects, the total momentum is the sum of the momentum for each link

$$\mathbf{p} = \sum_{i=1}^{n} \mathbf{p}_i = \sum_{i=1}^{n} m\mathbf{v}_i. \tag{3.26}$$

An important property of the momentum of a closed system, is that it is conserved despite collisions within the system. This means that if the snake robot collides with an obstacle, the momentum of the snake robot and obstacle should be unchanged. In this project, the obstacles neither have any mass nor velocity and hence no momentum. Thus, the only momentum in the employed model is that of the snake robot.

3.3 Computed torque control

The content of this section is taken from Chapter 11 of Modern Robotics [8].

3.3.1 PD control

A common feedback controller is linear proportional-derivative control, or PD-control:

$$\tau = \mathbf{K_p} \mathbf{q_e} + \mathbf{K_d} \dot{\mathbf{q}}_\mathbf{e}. \tag{3.27}$$

The control gains $\mathbf{K_p}$ and $\mathbf{K_d}$ are positive diagonal matrices. The proportional gain $\mathbf{K_p}$ acts as a virtual spring that tries to reduce the position error $\mathbf{q_e} = \mathbf{q_d} - \mathbf{q}$, where $\mathbf{q_d}$ is the desired joint angles. The derivative gain $\mathbf{K_d}$ acts as a virtual damper that tries to reduce the velocity error $\dot{\mathbf{q}}_\mathbf{e} = \dot{\mathbf{q}}_\mathbf{d} - \dot{\mathbf{q}}$.

Substituting the PD control law into the dynamics (3.13) yields

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{K_p}(\mathbf{q_d} - \mathbf{q}) + \mathbf{K_d}(\dot{\mathbf{q}}_\mathbf{d} - \dot{\mathbf{q}}). \tag{3.28}$$

The damping ratio ζ and natural frequency ω_n of this system is given as:

$$\zeta = \frac{\mathbf{K_d}}{2\sqrt{\mathbf{K_p}\mathbf{M}(\mathbf{q})}} \quad \text{and} \quad \omega_n = \sqrt{\frac{\mathbf{K_p}}{\mathbf{M}(\mathbf{q})}}. \tag{3.29}$$

$\zeta = 1$ yields a critically damped behaviour, while larger values yield an overdamped behaviour. Furthermore, $\mathbf{K_p}$ should be chosen as high as possible if fast response is desired.

3.3.2 Feedforward control

Another strategy for trajectory following is to use the model of the robot's dynamics (3.13) to proactively generate torques instead of waiting for errors. The feedforward torque is calculated as

$$\tau = \tilde{\mathbf{M}}(\mathbf{q_d})\ddot{\mathbf{q}}_\mathbf{d} + \tilde{\mathbf{C}}(\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}), \tag{3.30}$$

where the model is perfect if $\tilde{\mathbf{M}}(\mathbf{q}) = \mathbf{M}(\mathbf{q})$ and $\tilde{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$.

3.3.3 Feedforward and feedback linearization

As no model of the robot and environment will be perfect, it is not sufficient to use pure feedforward control. Combining the PD controller (3.27), with a proper choice of PD gains, and a feedforward term (3.30) will ensure exponential decay of the trajectory error (not just setpoint error).

Since $\ddot{\mathbf{q}}_\mathbf{e} = \ddot{\mathbf{q}}_\mathbf{d} - \ddot{\mathbf{q}}$,

$$\ddot{\mathbf{q}} = \ddot{\mathbf{q}}_\mathbf{d} + \mathbf{K_d}\dot{\mathbf{q}}_\mathbf{e} + \mathbf{K_p}\mathbf{q_e}. \tag{3.31}$$

Substituting (3.31) into the robot dynamics (3.13) gives the computed torque controller, also known as feedforward plus feedback linearizing controller:

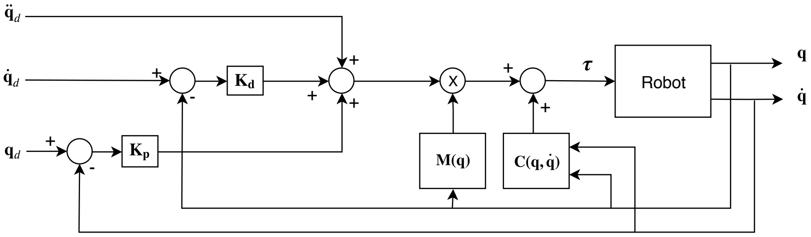


Figure 3.3: Computed torque control block diagram

$$\boldsymbol{\tau} = \tilde{\mathbf{M}}(\mathbf{q})(\ddot{\mathbf{q}}_\mathbf{d} + \mathbf{K_d}\dot{\mathbf{q}}_\mathbf{e} + \mathbf{K_p}\mathbf{q_e}) + \tilde{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}). \tag{3.32}$$

A block diagram of the resulting control (3.32) is shown in Figure 3.3.

3.4 Obstacle aided locomotion (OAL)

Instead of avoiding physical contact between the robot and obstacles, obstacle aided locomotion aims at profiting from it by using the obstacles as push-points to propel itself forward. OAL was first introduced by Transeth et al. in 2008 [2]. The motivation behind this method was based in the ability of biological snakes to utilize irregularities in the terrain for more efficient locomotion.

Liljebäck et al. [10] describe two major challenges related to OAL:

1. It is unknown in advance when and where the snake robot will make contact with its environment.
2. The development of a strategy for adjusting the shape of the robot so that forward propulsion is achieved in any given contact situation.

Furthermore, the following hypothesis is stated in [10].

> *Obstacle-aided snake robot locomotion can be achieved by producing body shape changes where the links in contact with obstacles are rotated so that the components of the contact forces in the desired direction of motion are increased.*

Holden et al. [13] address the second challenge by formulating an optimization problem that seeks to minimize energy consumption while achieving propulsion along a user-defined desired path. The output of this optimization is the optimal motor torque inputs. In addition to a user-defined path, this method assumes that the desired link angles at the obstacles are given.

Bayraktaroglu et al. [14] mentions that only the trajectory of the leading link should be arbitrarily determined. Moreover, Bayraktaroglu et al. [14] states that in a steady smooth motion, it must be computed as a function of available push-points for the next contact, and the desired position and orientation of the following links are those that mimic the motion of the leading link.

3.5 Hybrid position/force control (HPFC)

The goal of the snake robot is to push against obstacles in a fashion that yields forward propulsion along a path. Consequently, the robot will have to curve itself along the path whilst applying a force to the obstacles considered advantageous. The behavior of the robot has to comprise with the constraints arising from the contact, which further motivates the use of hybrid position/force control (HPFC).

HPFC is not a control method per se, but rather a method for determining when and in which directions force or motion control should be applied. It is desired to control motion along the unconstrained motion directions and force along the constrained motion directions. Different approaches to this problem exist. One is the use of selection matrices, introduced by Raibert and Craig et al. [3]. The disadvantage of this approach is that the directions in which force and motion should be controlled has to be recalculated for every step, and is no simple procedure. In another approach, introduced by West and Asada [4], two projection matrices are used as filters in joint space to automatically select between position- and force controlled vectors. The rest of this section covers this method and is based on the paper of West and Asada [4].

3.5.1 Constrained motion

Like mentioned above, velocity and force can be controlled in the directions in which they are not constrained. The end effector space is divided into two orthogonal domains, a position domain and a force domain, which are complementary to the directions of the corresponding constraints at the end effector. If there is contact with the environment, motion cannot be controlled freely. On the other hand, if there is no contact, there is no direction in which the robot can apply a force and the robot is force constrained. Ergo, the force and motion control directions do not overlap and the domains are orthogonal. This means that position and force can be controlled independently and arbitrarily in these domains.

The following relationships are known from Sections 3.1.1 and 3.2.1.

$$\mathbf{v} = \mathbf{J}\dot{\mathbf{q}}, \quad \boldsymbol{\tau} = \mathbf{J}^T\mathbf{f} \tag{3.33}$$

An important observation is that constraints due to contact with the environment are constraints due to a closed kinematic chain. In the snake robot case there might not always be two points in contact with the environment. It is however possible to define a virtual closed kinematic chain where the robot is connected to the base with the virtual joint variables x_0, y_0 and ϕ_1. A separate Jacobian is calculated for each closed kinematic chain, as explained in 3.1.1. Since the motion is constrained at a contact point, the relationships

Relationship (3.34) comes from the motion being constrained at a contact point.

$$\dot{\mathbf{v}}_{\mathbf{ci}} = \mathbf{J}_{\mathbf{ci}}\dot{\mathbf{q}} = \mathbf{0} \tag{3.34}$$

The solution to (3.34) can be proven to be

$$\dot{\mathbf{q}} = (\mathbf{I} - \mathbf{J}_{\mathbf{ci}}^{+}\mathbf{J}_{\mathbf{ci}})\mathbf{y}, \tag{3.35}$$

where $\mathbf{y}$ can be an arbitrary vector, as it will yield zero end effector motion. Furthermore, since the matrix $\mathbf{J}_{\mathbf{ci}}$ might be non square, the pseudo inverse $\mathbf{J}_{\mathbf{ci}}^{+}$ is used. For a closed kinematic chain, the work done at the end of the chain must also be zero. Therefore, the sum of the work done by each of the joints must be zero:

$$\boldsymbol{\tau}^{T}\dot{\mathbf{q}} = \boldsymbol{\tau}^{T}(\mathbf{I} - \mathbf{J}_{\mathbf{ci}}^{+}\mathbf{J}_{\mathbf{ci}})\mathbf{y} = \mathbf{0}. \tag{3.36}$$

(3.36) has the general solution

$$\tau = (\mathbf{J}_{\mathbf{ci}}^{+}\mathbf{J}_{\mathbf{ci}})^{\mathbf{T}}\mathbf{z}, \tag{3.37}$$

where $\mathbf{z}$ can be an arbitrary vector.

The allowable motion is now characterized by $[\mathbf{I} - \mathbf{J}_{\mathbf{ci}}^{+}\mathbf{J}_{\mathbf{ci}}]$ and the allowable forces by $[\mathbf{J}_{\mathbf{ci}}^{+}\mathbf{J}_{\mathbf{ci}}]^{\mathbf{T}}$. These matrices are orthogonal projectors in joint space onto the allowable position and force variations respectively. A further explanation of this result is given in Chapter 5 of [4]. The projectors will be abbreviated to

$$ {}_{ap}^{j}\mathbf{P} = [\mathbf{I} - \mathbf{J}_{\mathbf{ci}}^{+}\mathbf{J}_{\mathbf{ci}}] \quad \text{and} \quad {}_{af}^{j}\mathbf{P} = [\mathbf{J}_{\mathbf{ci}}^{+}\mathbf{J}_{\mathbf{ci}}]^{\mathbf{T}} = [\mathbf{I} - ({}_{ap}^{j}\mathbf{P})^{\mathbf{T}}]. \tag{3.38}$$

The superscript j denotes joint space, and ap and af stand for allowable positions and allowable forces respectively. It can be observed that these projection matrices project onto the nullspace of the respective constraint directions. This can further be related to the concept of task priority, in which tasks with lower priority are performed in the null-space of higher priority tasks [15].

3.5.2 Multiple constraints

If there are several contact points, projection matrices are calculated for each constraint, and the final projection matrices are found by taking the union and intersect of the different ${}_{af}^{j}P$ and ${}_{ap}^{j}P$ respectively.

3.5.3 Passive joints

The constraints (contact points) are modeled as virtual joints, described in 3.1.1. These joints are always passive, and since the corresponding forces are uncontrollable, they are always zero. To satisfy the additional constraint imposed by the passive joints, one can use a diagonal matrix **A** with ones on the diagonal indicating active joints. Another approach is to control the contact force at the contact point to satisfy the requirement that the force in the passive joints is zero.

3.5.4 Dynamic HPFC

This section briefly explains the dynamic hybrid control method based on the paper of Yoshikawa [5].

For describing the dynamics of the constraint manipulator, the constraints at the end effector are described by a set of constraint hypersurfaces before the equations of motion are derived. A given end effector constraint is expressed by a set of m hypersurfaces:

$$p_i(\mathbf{r}) = 0, \quad i = 1, 2, ..., m, \tag{3.39}$$

where $\mathbf{r}$ is the end effector position in a fixed frame. Furthermore, the relation between the joint variables $\mathbf{q}$ and the end effector position r is given by

$$\mathbf{r} = \mathbf{c}(\mathbf{q}). \tag{3.40}$$

Differentiating $\mathbf{c}(\mathbf{q})$ with respect to $\mathbf{q}$ gives the familiar Jacobian matrix $\mathbf{J}$.

Differentiating (3.40) with respect to time gives

$$\mathbf{E_F}\dot{\mathbf{r}} = 0, \tag{3.41}$$

where $\mathbf{E_F}$ consists of the unit normal vectors to the hypersurfaces (3.39).

The force exerted on the constraint surface by the end effector in the base frame b is denoted $\mathbf{f}^b \in \mathbb{R}^6$. Since the method assumes no friction between the surface and effector, then from the principle of virtual work $\mathbf{f}^b$ satisfies

$$\mathbf{v}^T\mathbf{f}^b = 0, \tag{3.42}$$

where $\mathbf{v}$ is the end effector velocity $\mathbf{T}\dot{\mathbf{r}}$. (3.41) can thus be written

$$\mathbf{E_F}\mathbf{T}^{-\mathbf{1}}\mathbf{v} = 0. \tag{3.43}$$

Eventually, the force $\mathbf{f}^b$, given (3.42) and (3.43), can be written

$$\mathbf{f}^b = \mathbf{E_F}\mathbf{T}^{-\mathbf{1}}\mathbf{f}_F, \tag{3.44}$$

where $\mathbf{f}_F \in \mathbb{R}^m$ is an unknown vector. It can be shown that the vector $-\mathbf{f}_F$ takes the same value as the Lagrange multiplier $\boldsymbol{\lambda}$ of the the Lagrange equations of motion for the manipulator under the following constraint:

$$\tilde{p}_i(\mathbf{c}(\mathbf{q})) = 0, \quad i = 1, 2, ..., m, \tag{3.45}$$

where $\tilde{p}_i(\mathbf{r}) = p_i(\mathbf{r})/||dp_i(\mathbf{r})/d\mathbf{r}||$.

The dynamics can now be expressed as

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{g}(\mathbf{q}) = \boldsymbol{\tau}_m + \mathbf{J}^{\mathbf{T}}(d\tilde{\mathbf{p}}/d\mathbf{r})^T \boldsymbol{\lambda}, \tag{3.46}$$

where $\boldsymbol{\tau}_m$ is the joint driving force. The left hand side of the equation is the general manipulator dynamics.

3.5.5 HPFC for snake robots

The joint torques of the robot should be calculated in a manner that allows for the robot to apply forces in certain directions while simultaneously positioning itself along the path. In order to find these torques, it is necessary to know in which direction the mentioned actions are allowable.

A snake robot lying alongside an obstacle is able to move along the obstacle. It is however unable to move through the obstacle. Hence, the direction in which the obstacle lies, limits the allowable position space. On the contrary, the allowable force space is restricted to the same direction. This is also sensible, as it makes no sense to attempt to apply a force in the direction of free space. Such an attempt would solely result in an uncontrolled acceleration of the robot joints.

The directions in which the robot can apply forces to obstacles is referred to as the allowable force space $\mathbb{F}$, and the directions in which it can move freely is referred to as the allowable position or motion space $\mathbb{M}$. The idea of Raibert and Craig et al. [3] was that these spaces both make out the subspaces of a bigger task space $\mathbb{T}$, which in the snake robot case is propulsion along a predefined path. The two subspaces are orthogonal, i.e. $\mathbb{T} = \mathbb{F} \cup \mathbb{M}$, $\mathbb{F} \perp \mathbb{M}$.

Stavdahl [1] introduces a ternary decomposition of the task space based on the fact that not all force interaction between the robot and the obstacles will yield propulsion. In some robot configurations, the reaction forces from the obstacles might even lead to a lock and increasing forces between the rigid bodies that can eventually lead to the robot breaking. The reaction forces illustrated in Figure 3.4 have no component that would push the robot in the forward direction. The robot could however bend its links and thus change its shape. Consequently it would be able to apply a torque yielding propulsion.

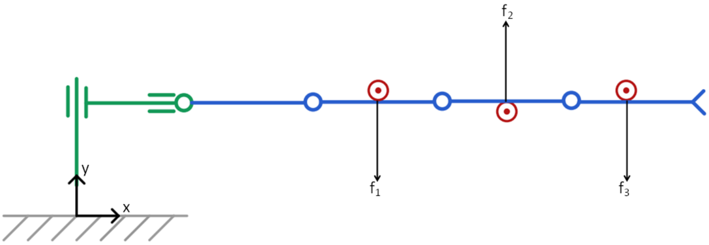


Figure 3.4: Configuration of the snake robot and obstacles yielding no propulsion

In other words there is a subspace of the force space which will not contribute to propulsion. Stavdahl [1] denotes this as the **constraint space** $\mathbb{C}$. The subspace within which forward motion of the head is achieved is denoted the **propulsion space** $\mathbb{P}$. The remaining subspace, the **shape space** $\mathbb{S}$, is the space in which the joint torques simply change the shape configuration of the robot.

Finally, knowledge about the specified spaces may be exploited in planning of a path from one point to another. In particular, it can be used to design an optimal path that maximizes the propulsion space. When the corresponding optimal forces are known, HPFC can be used to realise these forces while adjusting the shape of the robot to the path.

3.6 Projection onto path

In order for the robot to adjust its shape to fit the desired path, its controller is dependent on knowing the joint angles that fulfill this task. The desired joint angles are found by projecting the joints of the snake onto the predefined path. It is assumed that the 2D path is known at all times. Since the path has no width it is possible to simply find the shortest distance from a joint to the path to determine where the joint *should have* been. This distance, commonly known as the cross track error, is called BC and is illustrated in Figure 3.5. The path can be discretized to consist of points along the lines and curves defining the path.

The shortest distance is chosen by numerically comparing all the line segments from the joint to a piece of the path seen as relevant based on the position of the robot. When this point is known, the deviation angle θ can be determined by trigonometrical properties. The distance AB will equal the length of a link l, and the points A and B are respectively the positions of the preceding joint and joint to be projected. The distance AC is now called a and the distance BC is called b. The relationship in 3.47 is given from the law of cosine.

$$\cos(\theta) = \frac{l^2 + a^2 - b^2}{2lb}. \tag{3.47}$$

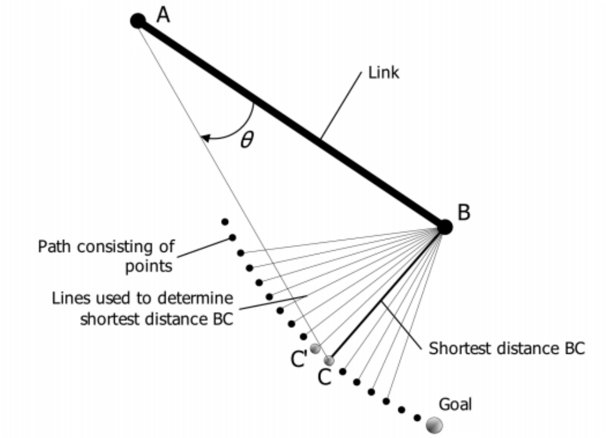

A
Link
θ
B
Path consisting of points
Lines used to determine shortest distance BC
Shortest distance BC
C'
C
Goal

Figure 3.5: Shortest distance from joint to path [16]

Note that this expression will be invalid if the shortest distance from the joint to the path is zero, meaning the joint is already on the path. There is however no need to perform a projection if there is no deviation from the path. In the case of very precise calculation, the joint and the path will never overlap completely. One can however introduce a very small width around the path in which the joint is considered to be on the path. This is equivalent to the threshold introduced in [16], where an angle deviation θ lower than a given threshold is ignored.

Because no explicit information about which side (left/right) of the path the joints are lying on is assumed, the sign of θ is unfamiliar. The easiest way to solve this in a computer program is simply comparing the result of rotation around A with both the positive and negative angle options. The link AB is rotated using the rotation matrix $\mathbf{R_z}$, expressed in (3.3). The resulting projected points are thus

$$C^{+} = A + \mathbf{R_z}(\theta)AB \quad \text{and} \quad C^{-} = A + \mathbf{R_z}(-\theta)AB. \tag{3.48}$$

The angle with the corresponding point closest to C is finally chosen and added to the actual angle of the joint to obtain the new desired angle.

3.7 Contact detection

Before the consequence of an interaction is calculated, the point of contact (if any) has to be determined. This is done by projecting the obstacle point onto the closest link. The distance from the obstacle to the projected point is then compared to the safety radius r_{safety} around each obstacle. This safety radius is to avoid the scenario in which the robot in a discrete time step moves to the opposite side of the obstacle point. Consequently, the relation in (3.49) is obtained.

$$\text{contact} = \begin{cases} 1 & \text{if } pc \leq r_{safety} \\ 0 & \text{else} \end{cases} \tag{3.49}$$

Figure 3.6 shows a case in which contact is established. If link i is considered in contact with an obstacle, the distance $d_{c,i}$, which is a generalized coordinate introduced in 3.1.1, equals $|AP|$.

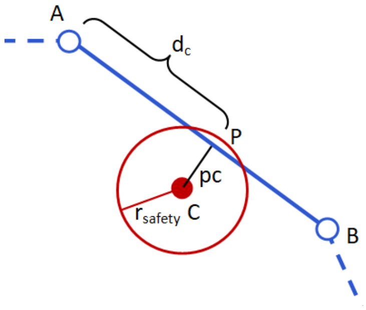

A
d_c
P
pc
r_{safety}
C
B

Figure 3.6: Robot link and obstacle with safety radius

Chapter 4

The simulator

The purpose of the simulator is to visualize the movement of a simple snake robot model progressing according to a predefined path and obstacles in the environment. Because the tested concepts are rather experimental and go in the direction of a proof-of-concept, the whole simulator is developed from scratch in MATLAB. The process of coding the simulator has also been a great resource for studying these concepts and the general theory of snake robots. The limitations coming from the experimental approach are presented below.

This chapter also aims at describing how the simulator is structured and can be used, as well as explaining particular code snippets in more detail. Moreover, it provides an overview of how the mathematical background is applied.

4.1 Simplifications and limitations

Expressing the complete dynamics of the system includes explicitly finding the part of the joint torques that are not directly applied to the robot joints, but rather a force acting on the external obstacles. In other words, the torque would have to be separated in a component belonging to the constraint or propulsion space and a component that belongs to the shape space (see 3.5.5). The relationship between the constraint or external torque is given in (3.24). However, both the force $\mathbf{f}_{c,i}^{b}$ and torque $\boldsymbol{\tau}_c$ are unknown. Additionally, solving the equation with respect to a hypothetically given $\boldsymbol{\tau}_c$ would not necessarily yield a unique solution of $\mathbf{f}_{c,i}^{b}$.

This challenge, combined with a strict time schedule, has led to a simplification in the development of the simulator. In the program, the movement of the robot is first calculated under the assumption that no obstacle is present. To account for this, the interaction forces are implicitly considered through mapping to the allowable position space. This means that if the robot is in contact with the environment, the joint velocities are influenced accordingly and thus also the joint angles. The consequences of the simplification are discussed in Chapter 6.

Like in any computer simulation, the time is discretized. This leads to inaccuracies, especially for fast motions. The sampling time may be decreased to minimize this effect.

Furthermore, the projection onto the desired path assumes that the robot is sufficiently close to the path to get back to it. The calculated desired angles for joints very far away from the path might therefore not lead to advantageous behaviour. The projection also assumes that the links of the robot are chronologically aligned along the path. This means that if the robot is for instance curled up, the projection will fail to align the robot. One last point to note is that the desired angle of the tail is not specified by the method. This property is pointed out in 5.1.2 and discussed further in 6.2.

4.2 Program flow

The program flow is illustrated in the diagram Figure 4.1. The blue rectangles represent functions, and the data connected to the outgoing arrows are computed output variables. From the diagram, it is noticeable that the program runs in a loop after initialization, constantly controlling the robot towards the path while adapting to its environment (the obstacles).

Initialization is performed based on the given snake robot and obstacle configurations. Furthermore, the joint acceleration $\ddot{\mathbf{q}}$ of the robot is found by solving the dynamics equation (3.14), where the joint torque $\boldsymbol{\tau}$ is calculated by the computed torque control (3.32). The control reference is based on deviation from the desired trajectory (see Section 3.6). Joint velocities $\dot{\mathbf{q}}$ and angles $\mathbf{q}$ are deduced from $\ddot{\mathbf{q}}$ using Euler integration.

Whenever the robot is in contact with one or more obstacles, the joint velocities are projected onto the allowable position space using the intersect of all ${}^{j}_{ap}P$, explained in 3.5.1.

4.3 User guide

The user can adjust parameters related to the controller, the specifications of the snake robot, obstacles in the environment and the desired path the robot

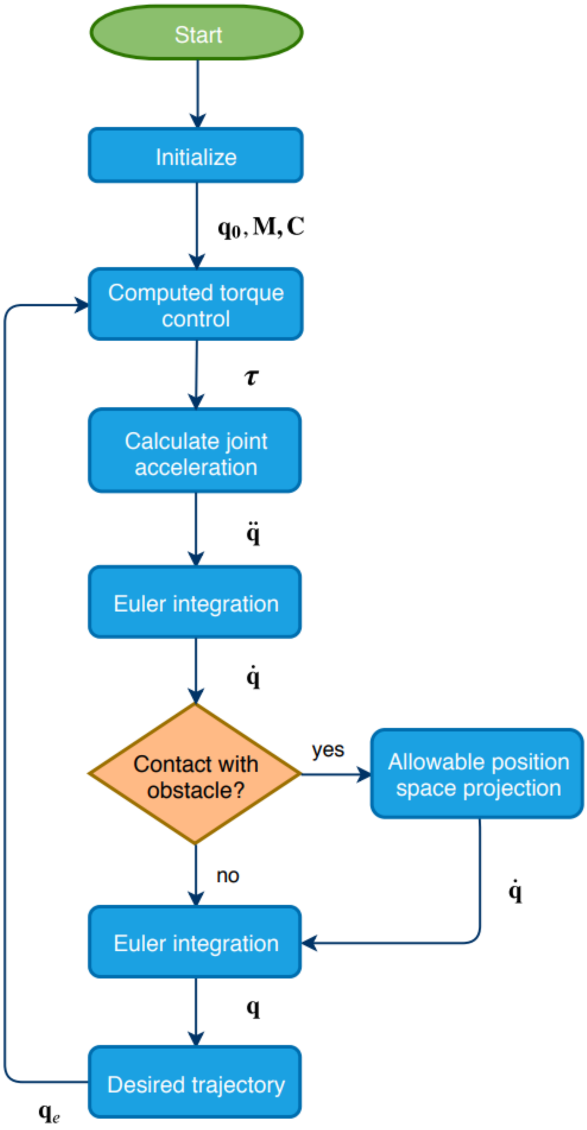

Start
Initialize
$\mathbf{q}_0, \mathbf{M}, \mathbf{C}$
Computed torque control
$\boldsymbol{\tau}$
Calculate joint acceleration
$\ddot{\mathbf{q}}$
Euler integration
$\dot{\mathbf{q}}$
Contact with obstacle?
yes
Allowable position space projection
$\dot{\mathbf{q}}$
no
Euler integration
$\mathbf{q}$
Desired trajectory
$\mathbf{q}_e$

Figure 4.1: Program flow diagram

should attempt to follow. an overview of the user definable variables and where they can be adjusted is given in Table 4.1.

Once the variables are adjusted in the respective files, the `initialization.m` script can be run. For a specific configuration, this initialization only has to be run once for every MATLAB workspace. The simulation can then be run for a desired number of times. Variables that do not change the dynamical properties of the robot can be redefined without running the whole initialization over.

Figure 4.2 gives a description of the components in the visual part of the simulation.

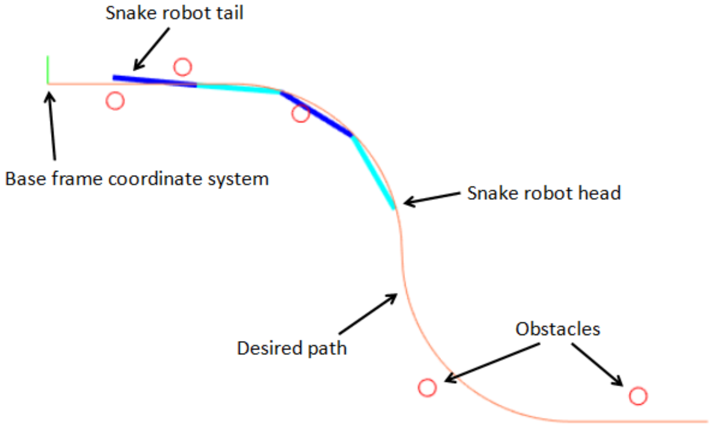

Snake robot tail
Base frame coordinate system
Snake robot head
Desired path
Obstacles

Figure 4.2: Visual output of simulation

Parameter description	**Parameter name**	**Filename**	**Datatype**	**Unit**
Number of links	`n`	`init_snake.m`	Integer	
Link length	`l`	`init_snake.m`	Float	m
Link mass	`m`	`init_snake.m`	Float	kg
Initial joint angles	`q0(1:n)`	`init_snake.m`	Array of floats	rad
Initial position	`q0(n+1:n+2)`	`init_snake.m`	Array of floats	rad
Number of obstacles	`num_obstacles`	`init_obstacles.m`	Integer	
Obstacle coordinates	`obstacle_coords`	`init_obstacles.m`	2D array of floats	m
Obstacle safety radius	`obstacle_radius`	`init_obstacles.m`	Float	m
Proportional control gain	`kp`	`init_control.m`	Float	
Damping ratio	`zeta`	`init_control.m`	Float	
Torque saturation limit	`max_tau`	`init_control.m`	Float	Nm
Number of individual functions defining the path	`num_sections`	`init_path.m`	Integer	
Intersection point of path functions along x	`section_partition`	`init_path.m`	Array of floats	m
Functions defining the path	`curve`	`init_path.m`	Set of functions	m

Table 4.1: User adjustable simulator parameters

4.4 Program code summary

The whole simulator is made, and to be executed, in MATLAB. The symbolic toolbox [17] has been used to generalize the code and make it adaptable to an n-link snake robot. The whole code can be provided upon request.

4.4.1 Dynamics functions

The dynamics, more particularly the matrices **M** and **C** from (3.13), are generated symbolically in the script `init_dynamics.m`. As a result of the use of symbols, the matrices can be used as functions with the joint angles and velocities as input parameters.

4.4.2 Kinematics functions

The script `init_kinematics.m` generates the transformation matrices from (3.2), defining the relation from base to every link endpoint. These matrices are – equivalent to the dynamic matrices from last section – expressed with symbols. Additionally, functions for generating the rotation matrix around the z-axis, `Rz_func(theta)`, and translation matrix in the x and y direction, `D_func(x_dist, y_dist)` are defined.

4.4.3 Contact Jacobian function

The symbolic expression of the contact Jacobians is calculated in `init_contact_jacobians.m`. A selection of the code is given and explained below.

- **Lines 2-8:** The position of the contact point in the base frame expressed by the joint variables is extracted from the homogeneous transformation matrix (see Section 3.1).

- **Lines 9-12:** Partial differentiation of the position of the contact point with respect to the generalized coordinates to obtain the contact Jacobian.

- **Line 14:** All contact Jacobians are stacked in a 3-dimensional matrix.

The variable `q` represents the vector of generalized coordinates, and `Rz_func(theta)` and `D_func(x_dist, y_dist)` are previously defined functions (see 4.4.2).

```
 1  for link = 1:n % For every link an obstacle can be in contact with
 2      T = D_func(q(n+1),q(n+2));
 3      for curr_link = 1:link-1
 4          T = T*Rz_func(q(curr_link))*D_func(1,0);
 5      end
 6      T = T*Rz_func(q(link))*D_func(q(n+2+link));
 7      x = T(1,4);
 8      y = T(2,4);
 9      for i = 1:N
10          Jc(1,i) = diff(x, q(i));
11          Jc(2,i) = diff(y, q(i));
12      end
13      Jc = simplify(Jc);
14      all_Jc(:,:,link) = Jc;
15  end
```

4.4.4 Projection function

The projections are calculated in the function `calc_projections.m` based on the contact Jacobian function Section 4.4.3. In the following code snippet, contact on link `i` is already established and the corresponding projection matrices are to be found.

- **Line 2:** The variable `q(n+2+i,k)` corresponds to the generalized coordinate $d_{c,i}$ (see (3.8)).
- **Lines 3-4:** The contact Jacobian with current $\mathbf{q}$-values from simulation.
- **Lines 5-6:** Projection matrices from (3.38).
- **Lines 7-8:** Projection matrices are stacked.

```
1  l_to_obs = ap; % Distance from joint to obstacle point projection:
2  q(n+2+link,k) = l_to_obs;
3  all_Jc = Jc_func(q(:,k)’);
4  Jc(:,:,i) = all_Jc(:,:,i); % Extract the relevant contact Jacobian
5  P_af = (pinv(Jc(:,:,i))*Jc(:,:,i))’;
6  P_ap = eye(N) - pinv(Jc(:,:,i))*Jc(:,:,i);
7  S_P_af = [S_P_af P_af];
8  S_P_ap = [S_P_ap P_ap];
```

Seeing as there might be more than one contact point, several projection matrices are calculated. These matrices are combined by taking the union and intersect of the allowable force projectors and allowable position projectors respectively (described in 3.5.1). The logic of the union and intersect functions are borrowed from Øyvind Stavdahl and presented below.

```
% Union of all P_af
P_af = S_P_af*pinv(S_P_af);
% Intersect of all P_ap
P_ap = S_P_ap(:,1:N)*pinv(S_P_ap(:,1:N));
for i = 2:num_contacts
    P_temp = S_P_ap(:,N*(i-1)+1:N*i);
    P_temp = P_temp*pinv(P_temp);
    P_ap = 2*(P_ap - P_ap*pinv(P_ap + P_temp)*P_ap);
end
```

4.4.5 Control

The following function calculates the joint torques from the computed torque control scheme in 3.3.3.

- **Line 3:** See (3.32).
- **Line 5:** Saturation based on the user-adjustable parameter `max_tau`.

```
1 function tau_control = computed_torque_control(M, C, q_e, qd_e)
2     qdd_ref = zeros(N,1);
3     tau_control = M*(qdd_ref + kd*qd_e + kp*q_e) + C;
4     % Restrict torque to actuated joints and limit the magnitude.
5     tau_control = saturate(tau_control);
6 end
```

4.4.6 Visualization

The robot simulation is performed by displaying a figure in which the robot links are redrawn for every time step based on the given robot data. The function in `visualize.m` takes care of this operation.

Chapter 5

Experiments

This chapter presents simulation scenarios chosen to demonstrate the performance and limitations of the simulator, as well show the concept of OAL.

A common configuration for all experiments is given in Table 5.1. The individual configurations are presented in the respective sections.

Description	**Variable name**	**Value**
Link length $[m]$	`l`	1
Link mass $[kg]$	`m`	1
Obstacle safety radius $[m]$	`obstacle_radius`	0.1

Table 5.1: Common simulation configuration for all cases

5.1 Obstacle-free environment

This section demonstrates how the robot behaves without the influence of obstacles in the environment. Special emphasis is placed on the the path alignment performance.

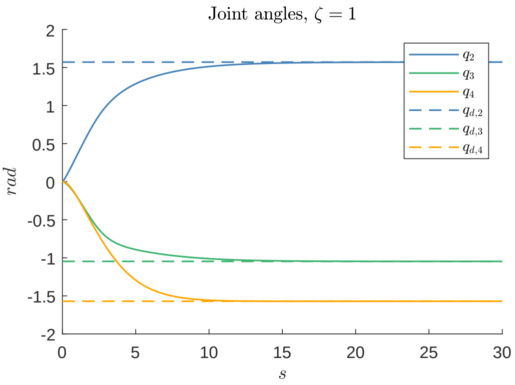
5.1.1 Single reference computed torque control

This experiment is to demonstrate the two different damping modes of the computed torque controller applied in the rest of the experiments. For simplicity, a static setpoint scenario is chosen. The variable configuration for the experiment is summarized in Table 5.2.

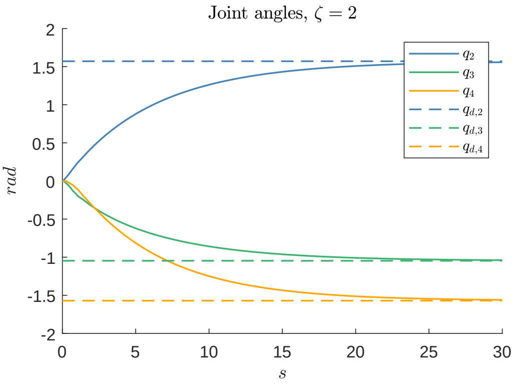
Description	**Variable name**	**Value**
Simulation time $[s]$	`simTime`	30
Simulation sample time $[s]$	`h`	0.01
Number of links	`n`	4
Joint angle setpoints $[rad]$	`q_ref`	$[0, \pi/2, -\pi/3, -\pi/2]$
Initial joint angles $[rad]$	`q_0(1:n)`	$[0, 0, 0, 0]$
Initial position $[m]$	`q_0(n+1:n+2)`	$[0, 0]$

Table 5.2: Simulation configuration for 5.1.1

Figure 5.1 shows the different damping-cases using the computed torque controller following joint angle references plotted with dashed lines. It should be noted that the joint angle $q_1 = \phi_1$ is not included as it is a virtual coordinate and not directly actuated.



(a) Critically damped



(b) Overdamped

Figure 5.1: Simulation demo - computed torque control

Because slow motions are desired to avoid abrupt movement, the overdamped configuration with $\zeta = 2$ is mainly used throughout the experiments. The underdamped case is not relevant for this project and thus left out.

The proportional gain is chosen to be $\mathbf{K_p} = 0.4\mathbf{I}$ in all cases. The relation in (3.29) with $\zeta = 1$ and $\zeta = 2$ yields $\mathbf{K_d} = 1.3\mathbf{I}$ and $\mathbf{K_d} = 2.5\mathbf{I}$ respectively.

5.1.2 Path alignment

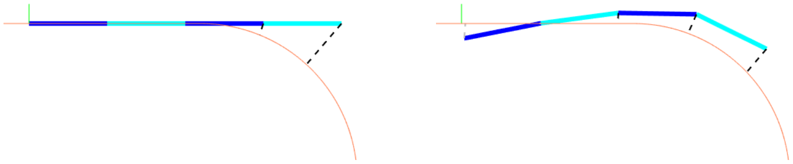
In this case, the robot is initialized with two different start positions before attempting to adjust to the same path (5.1), which consists of line segments and quadrants. The start positions together with remaining variable configurations are summarized in Table 5.3.

$$y(x) = \begin{cases} 0, & \text{if } x < 2.2 \\ \sqrt{4-(x-2.2)^2}+2, & \text{if } x \in [2.2, 4.2] \\ -\sqrt{4-(x-6.2)^2}+2, & \text{if } x \in [4.2, 6.2] \\ -4, & \text{if } x > 6.2 \end{cases} \tag{5.1}$$

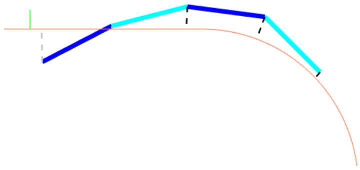
Description	**Variable name**	**Value**
Simulation time $[s]$	`simTime`	10
Simulation sample time $[s]$	`h`	0.001
Damping ratio	`zeta`	2
Number of links	`n`	4
Joint angle setpoints $[rad]$	`q_ref`	Given by the path projection method
Initial joint angles $[rad]$	`q_0(1:n)`	$[0, 0, 0, 0]$
Initial position $[m]$	`q_0(n+1:n+2)`	$[0, 0]$
Initial position part 1 $[m]$ Initial position part 2 $[m]$	`q_0(n+1:n+2)`	$[0, 0]$ $[0, 1]$

Table 5.3: Simulation configuration for 5.1.2

The Figures 5.2 and 5.3 show the start and end position of the adjustment in a 15 second interval. The dashed lines are connections between the joints and their respective projected points on the path. The lines for the first two joints are displayed in gray because they are disregarded in the computation of the reference angles. The path projection method simply regards endpoints of controllable links. Furthermore, the method is not considering that the snake robot should be stretched out along the path at all times. A repercussion of this will become clear in the experiment in 5.3.3.

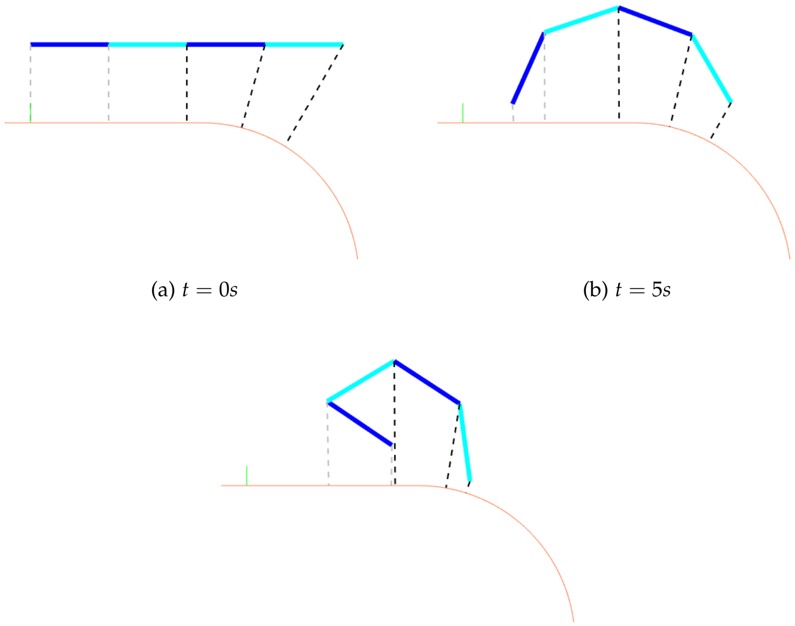


(a) $t = 0s$ (b) $t = 5s$



(c) $t = 10s$

Figure 5.2: Simulation demo - adjusting to path from nearby



(a) $t = 0s$

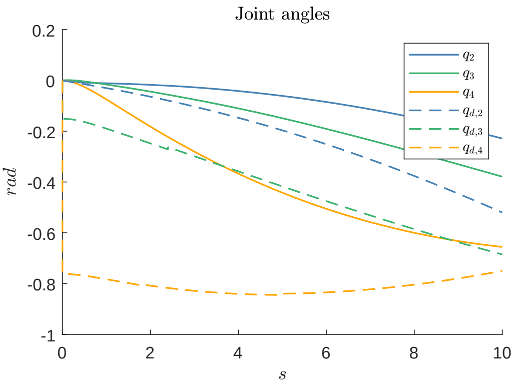
(b) $t = 5s$

(c) $t = 10s$

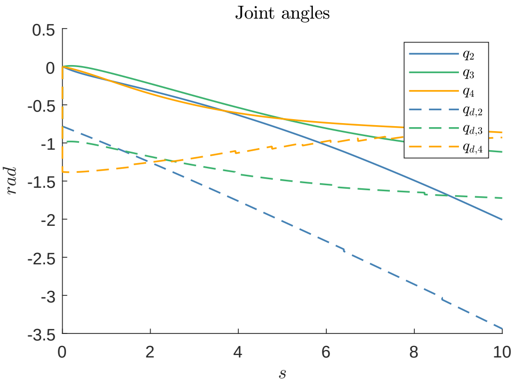
Figure 5.3: Simulation demo - adjusting to path from a distance

It is evident that path adjustment relies on a start configuration close to the path to be successful as well. However, none of the simulations are able to perfectly adjust to the path. An important reason for this is that the robot cannot change the position of its center of mass without friction or push-points (obstacles) in the environment.

The plots in Figure 5.4 show the joint angles of the controllable joints during the simulations. The reference angles are plotted with a dashed line. From the plots it can be observed that the end effector is very close to the path and the corresponding reference angles $q_{d,4}$ are almost met in both cases. Additionally, the reference angles of link 2 and 3 wish to bend the links downward to the path. This is logical, just not completely feasible. A consequence of this is that the robot too far from the path is curling up.



(a) Start at $(x_0, y_0) = (0, 0)$



(b) Start at $(x_0, y_0) = (0, -1)$

Figure 5.4: Joint angles for path adjustment with different starting configurations

5.2 Collision analysis

This section demonstrates the energy and momentum difference in the robot before and after being influenced by an obstacle. The experiment is performed for validity analysis of the interaction approximations, and the outcome is discussed further in Chapter 6.

5.2.1 Single obstacle interaction

For simplicity, only one obstacle is included in this experiment. Furthermore, the robot is not set to follow any joint angle references. Instead, a step input torque with magnitude $-0.5Nm$ is applied to the foremost joint for 0.2 seconds. The simulation configuration is given in Table 5.4.

Description	**Variable name**	**Value**
Simulation time $[s]$	`simTime`	3
Simulation sample time $[s]$	`h`	0.001
Damping ratio	`zeta`	2
Number of links	`n`	4
Initial joint angles $[rad]$	`q_0`	$[0,0,0,0]$
Initial position $[m]$	`q_0(n+1:n+2)`	$[0,0]$
Number of obstacles	`num_obstacles`	1
Obstacle position $[m]$	`obstacle_coords`	$(3.5,-0.3)$

Table 5.4: Simulation configuration for 5.2.1

The torque is chosen to make the foremost robot link collide with the obstacle so that the resulting change in energy and momentum can be observed.

The different positions of the robot to different times are illustrated in Figure 5.6. Furthermore, the energy (3.17), momentum (3.26) and applied torque is plotted in Figure 5.5.

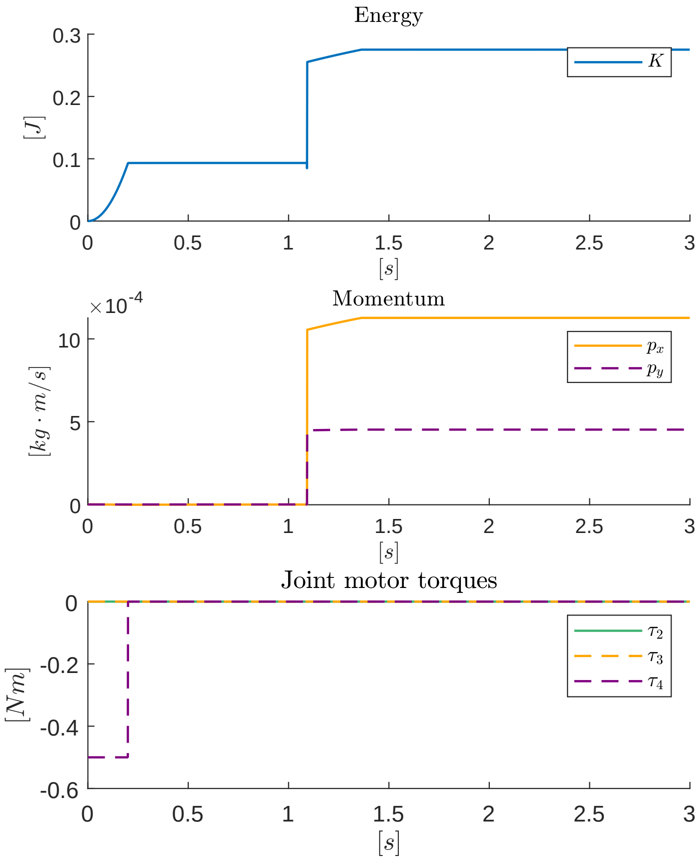


Figure 5.5: Energy, momentum and joint motor torques of robot for single collision scenario

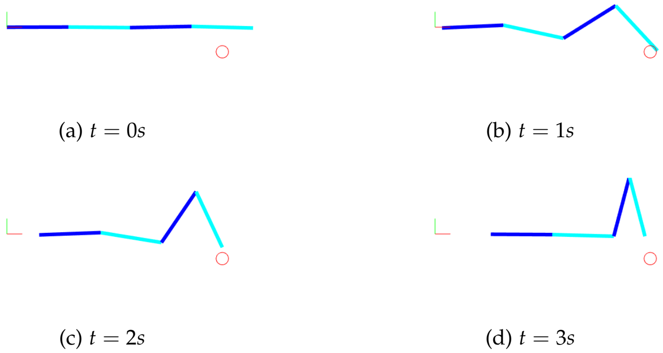

(a) $t = 0s$
(b) $t = 1s$
(c) $t = 2s$
(d) $t = 3s$

Figure 5.6: Simulation demo - interaction with single obstacle

The motor torque will naturally increase the energy of the robot for the time period in which it is applied. According to laws of physics, the energy level should then stay constant as long as no work is done on the robot and it is moving in a frictionless environment. The energy plot in Figure 5.5 confirms this, seeing as the energy stays constant between 0.2 to 1.1 seconds. When the robot then hits the obstacle, a clear change in energy is observed. The approximate time of this collision is further confirmed by the configuration of the robot at 1 seconds seen in part (b) of Figure 5.6. In reality, the robot would at this point loose energy to the obstacle. From the plot it can however be observed that the energy is increased until the robot link no longer is in contact with the obstacle. The total energy of the robot before the contact is $0.0933J$ and $0.2751J$ after the contact, meaning the energy is increased by 295%.

Furthermore, the momentum of the snake robot should be conserved despite the collision (see 3.2.2). However, it turns out that the momentum in x-direction increases by $1.1271 kg \cdot m/s$ and the momentum in y-direction increases by $0.4519 kg \cdot m/s$, violating the well known *law of conservation of momentum*.

It should be noted that the robot cannot change the position of its center of mass without an external force. Therefore, even though the links obtain velocities after the torque is applied, they cancel each other out in the respective directions. The moments in the two directions are determined by the velocities and accordingly unchanged by the torque, as seen in the momentum plot in Figure 5.5.

5.3 OAL simulation scenarios

5.3.1 Simple propulsion

This scenario aims at demonstrating the concept of OAL. The snake robot is simply set to bend its front joint to $-\pi/2$ while the other joints are to remain in a stretched out configuration. The simulator variable configuration can be seen in Table 5.5.

The setpoints are manually determined based on the knowledge that the bending link will collide with an obstacle in trying to obtain the desired angle. Hence, the link will apply a force to the obstacle underneath and consequently drag the whole robot body in the positive x (rightward) direction. A sequence of the movement is presented in Figure 5.7.

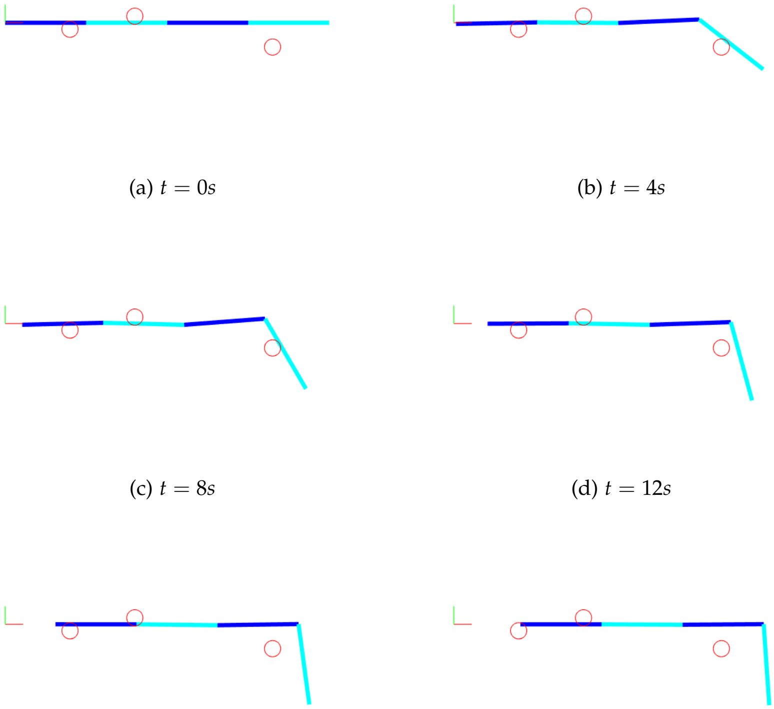
Description	**Variable name**	**Value**
Simulation time $[s]$	`simTime`	20
Simulation sample time $[s]$	`h`	0.001
Damping ratio	`zeta`	2
Number of links	`n`	4
Joint angle setpoints $[rad]$	`q_ref`	$[0, 0, 0, -\pi/2]$
Initial joint angles $[rad]$	`q_0`	$[0, 0, 0, 0]$
Initial position $[m]$	`q_0(n+1:n+2)`	$[0, 0]$
Number of obstacles	`num_obstacles`	3
Obstacle positions $[m]$	`obstacle_coords`	$(0.8, -0.08)$ $(1.6, 0.08)$ $(3.3, -0.3)$

Table 5.5: Simulation configuration for 5.3.1

The obstacles laying close to the rear links are positioned to allow the rest of the robot to stay flat. From Figure 5.7 it can be seen that the robot moves away from the obstacles towards the end without pushing against anything. This is a consequence of the modeled frictionless environment.

From the plots in Figure 5.8 it is clear that the collision with the lower obstacle takes place at approximately 3 seconds. At this point, the joint velocities are projected such that the front link does not cross the obstacle. Additionally, the preceding joints experience an offset as the front joint applies a torque that influences the whole robot.

A repercussion of the geometrical approximation of the contact forces becomes obvious in this experiment. The joint velocities in the emphasised time span in part (b) of Figure 5.8 admit a twitching behaviour as a result of the velocity projections. This circumstance is discussed in Chapter 6.



(a) $t = 0s$

(b) $t = 4s$

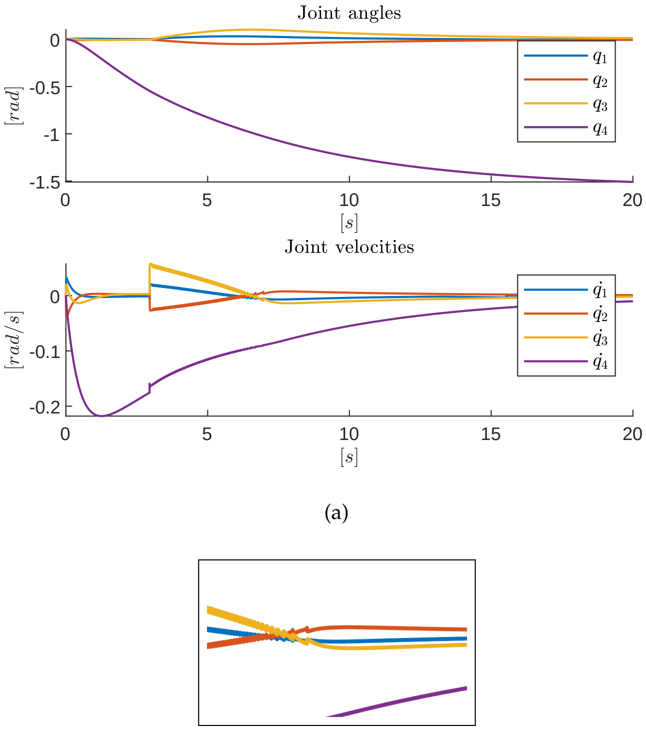
(c) $t = 8s$

(d) $t = 12s$

(e) $t = 16s$

(f) $t = 20s$

Figure 5.7: Simulation demo - propulsion with static joint setpoint



(a)

(b) Zoom of joint velocities in seconds 5-10

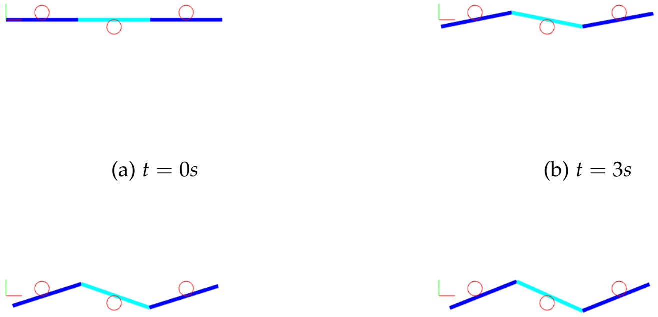
Figure 5.8: Joint angles and velocities for the single setpoint scenario

5.3.2 Obstacle interaction without propulsion

Seeing as propulsion requires a force in the respective direction, there are scenarios where a set of joint torques are insufficient for attaining this force. This scenario aims at illustrating the case where the forces applied work against each other in the direction of propulsion and the robot is simply deformed. The variable configuration for the simulation is presented in Table 5.6 and Figure 5.9 shows a sequence of the motion.

Description	**Variable name**	**Value**
Simulation time $[s]$	`simTime`	9
Simulation sample time $[s]$	`h`	0.001
Damping ratio	`zeta`	1
Number of links	`n`	3
Joint angle setpoints $[rad]$	`q_ref`	$[0, -\pi/3, \pi/3]$
Initial joint angles $[rad]$	`q_0`	$[0, 0, 0]$
Initial position $[m]$	`q_0(n+1:n+2)`	$[0, 0]$
Number of obstacles	`num_obstacles`	3
Obstacle positions $[m]$	`obstacle_coords`	$(0.5, 0.1)$ $(1.5, -0.1)$ $(2.5, 0.1)$

Table 5.6: Simulation configuration for 5.3.2



(a) $t = 0s$

(b) $t = 3s$

(c) $t = 6s$

(d) $t = 9s$

Figure 5.9: Simulation demo - no propulsion

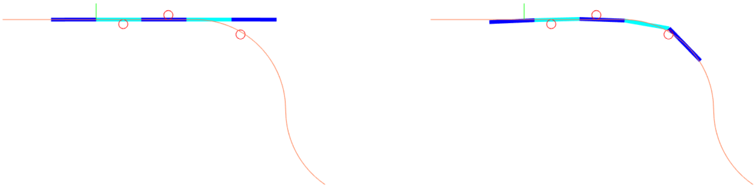
5.3.3 Unsuccessful propulsion attempt along path

This experiment demonstrates a case in which the simplifications made are preventing the robot from moving in the desired direction. The robot is at all times merely set to adjust itself to the path without any objective of propulsion. Just like in 5.3.1, the obstacles blocking the robot are the factor leading to propulsion. The configuration is summarized in Table 5.7, and the desired path (5.1) is the same one as in 5.1.2.

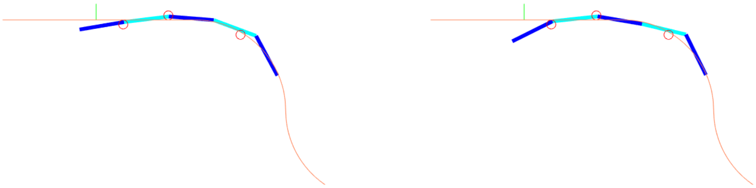
Description	**Variable name**	**Value**
Simulation time $[s]$	`simTime`	60
Simulation sample time $[s]$	`h`	0.005
Damping ratio	`zeta`	2
Number of links	`n`	5
Joint angle setpoints $[rad]$	`q_ref`	Given by the path projection method
Initial joint angles $[rad]$	`q_0`	$[0, 0, 0, 0, 0]$
Initial position $[m]$	`q_0(n+1:n+2)`	$[-1, 0]$
Number of obstacles	`num_obstacles`	3
Obstacle positions $[m]$	`obstacle_coords`	$(0.6, -0.1)$ $(1.6, 0.1)$ $(3.2, -0.33)$

Table 5.7: Simulation configuration for 5.3.3

The resulting problem is caused by the path projection method that disregards the positioning of the tail. When the tail bends over an obstacle, the resulting contact force pushes the robot in the wrong direction while the front link is pushing it in the right direction. The result is that the robot gets stuck. A sequence of the movement is presented in Figure 5.10.



(a) $t = 0s$ (b) $t = 20s$



(c) $t = 40s$ (d) $t = 60s$

Figure 5.10: Simulation demo - failed propulsion along path

5.3.4 Propulsion along path

This case is to illustrate how the robot can aid obstacles to propel along a predefined path. Since the robot is only position controlled based on deviation from the path, the obstacles are placed in a manner that supports both the desired shape and motion. The two obstacles in the middle are to keep the rear links on the path, while the obstacle in front is placed so that the robot can push against it and pull itself forward. In order for this to happen, the obstacle has been placed slightly on the path like in previous experiments, leading to a constant deviation from the path. The robot's desire to always stay as close to the path as possible makes it push against the obstacle.

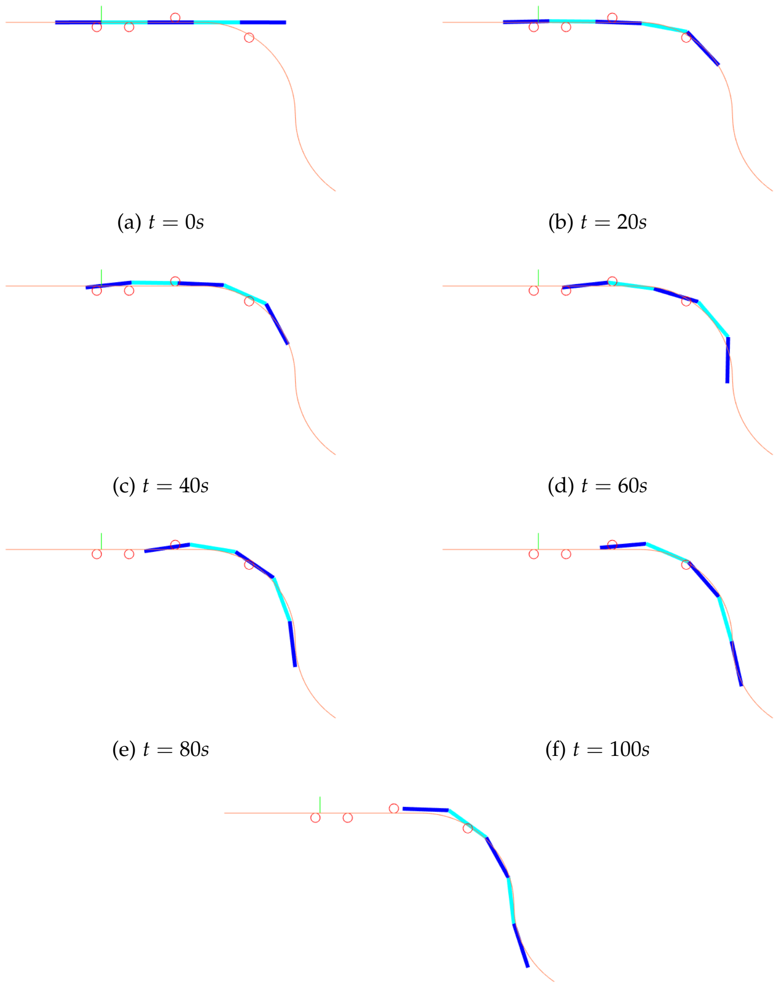
The leftmost obstacle is added based on the observations made in 5.3.3, where the tail deviated from the path and caused the robot to get stuck. The tail link will now be pushed back to the path when in contact with this obstacle. An important point to note is that the spacing between the two leftmost obstacles violates assumption 10 in 2.1, saying that every link is in contact with at most one obstacle. The simulator is however able to handle this scenario and compute contact Jacobians and projection matrices (see 3.5.1-3.5.2) for both contact points.

The variable configurations are presented in Table 5.8 and the desired path (5.1) is the same one as in 5.1.2.

Description	**Variable name**	**Value**
Simulation time $[s]$	`simTime`	120
Simulation sample time $[s]$	`h`	0.005
Damping ratio	`zeta`	1
Number of links	`n`	5
Joint angle setpoints $[rad]$	`q_ref`	Given by the path projection method
Initial joint angles $[rad]$	`q_0`	$[0,0,0,0,0]$
Initial position $[m]$	`q_0(n+1:n+2)`	$[-1,0]$
Number of obstacles	`num_obstacles`	4
Obstacle positions $[m]$	`obstacle_coords`	$(-0.1,-0.1)$ $(0.6,-0.1)$ $(1.6,0.1)$ $(3.2,-0.33)$

Table 5.8: Simulation configuration for 5.3.4

The movement is shown in Figure 5.11, where it can be seen that the robot follows the path as long as it has sufficiently many obstacles to push against. At 40 seconds the robot starts moving slightly through the radius of the third obstacle. This is an effect of the projection of the joint velocities.



(a) $t = 0s$

(b) $t = 20s$

(c) $t = 40s$

(d) $t = 60s$

(e) $t = 80s$

(f) $t = 100s$

(g) $t = 120s$

Figure 5.11: Simulation demo - propulsion along path

Chapter 6

Discussion

This chapter looks at the effects of simplifications and approximations made during the development of the simulator, both in regard to the dynamical model and in regard to the path following method. These effects, as well as the general results from the experiments are discussed and some improvements are proposed.

6.1 Effects of the contact behaviour simplification

The geometrical approximation made regarding the contact forces between the robot and the obstacles have been characteristic for the outcome of the simulations. From a physical perspective, it is obvious that projecting the velocities to an "allowable space" just based on the positioning of the obstacles differs from actual velocities after a collision. Simply projecting the velocities means that the velocity of the robot body will in some cases increase, and sometimes decrease, after contact with an obstacle. This does in turn mean altering the total energy of the system, which contradicts basic laws of physics under the assumption of frictionless contact. In the experiment in 5.2.1, the energy of the robot was almost tripled after contact with a single obstacle.

Additionally, it was observed that the momentum was not conserved through the collision, neither in the x nor y-direction. Although these observations do not make out a general quantification of the error, it is enough for recognizing that the modelling of interactions in the simulator are considerably off. A further consequence is also that the robot can inherit abrupt and unrealistic movements, but because the dynamics of the employed model have been very slow in all experiments, it has not been a dominating factor and the controller has been able to damp all fast motions. Lastly, it has to be mentioned that the velocity projections can in some cases contribute to additional propulsion of the robot as they introduce new energy to the system.

There are of course several methods that could avoid this simplification. One option is to implicitly define the forces as a part of the dynamical model, just like the forces between the joints in the robot are implicitly defined. The method of Yoshikawa [5], explained in 3.5.4, follows this approach. It does however only consider constraints on the end effector of the robot, and not on arbitrary links. The adjustment that has to be made in the snake robot case, is that the constraint hypersurfaces and forces onto these surfaces have to be defined for every contact point.

Another consequence of the geometrical approximation, is that the projection matrices only allow movement along obstacles. This is necessary for preventing the robot from moving through obstacles, but also prevents the robot from perpendicularly moving away from them. In other words, whenever a link comes in contact with an obstacle, it will stick to it until it has slid all the way along it. This strict positioning of the links can come in conflict with the controller and path projection. The only case in which a link "detaches" from the obstacle before sliding along it, is when it in a discrete time step is projected to a position where it no longer is considered in contact and fazed by the obstacle.

Yet another important remark, is that the program treats the cases in which a link is on the edge of the obstacle radius and within the obstacle radius equally. In particular, it considers both cases a point contact and disregards how close it is to the actual obstacle point. In 5.3.4 it is pointed out that the robot slides slightly through this radius, but it should be kept in mind that for the robot it is the same situation as sliding on the edge of the obstacle. However, it is still a fault that the robot ended up within the radius in the first place, but is most probably a result of the discrete nature of the simulator.

This discrete behavior is also the sole purpose of the obstacle radius, preventing links from jumping over obstacles in discrete time steps, and has in this regard proven to be a successful mechanism. Furthermore, it is the velocity rather than position of the robot that is changed with the projection. The robot might thus still be inside the obstacle radius at the next time step and the velocity is projected over again. This can in turn lead to inconsistent velocity projections and a twitching behaviour as seen in 5.1.1. Decreasing the radius would naturally also decrease this effect, but it is still crucial to keep the radius in discrete-time simulations. The effect can also be decreased with a smaller sample time, but likewise this is no permanent solution to the problem.

A workaround for the rigid definition of contact in which the robot is either completely in contact with the obstacle or not at all, could be introducing an elastic radius or force field around the obstacles, and thus damp the nonlinear effects of the interactions.

6.2 Review of the path alignment method

The method of finding the desired angles based on projection onto the path is not robust in cases where the links are far from the path (see experiment in 5.1.2). The simplest solution to this would be avoiding the projection of links that are very misplaced. A future, more advanced, solution would be redesigning the optimal path to overlap with the current position of the robot. In 5.1.2 it is pointed out that the method of finding the desired joint angles disregards the positions of the two first joints, or rather the position of the start- and end point of the tail link. Correcting this poor quality would improve the performance of the path following capability significantly. Situations like in 5.3.3, where the tail of the robot gets stuck, would then be avoided.

The desired joint angles from the path projection method are calculated separately for each link in the current program. A more robust and intelligent solution would be defining an objective function that considers the deviation from the path for all joints, and then find the set of desired joint angles minimizing the complete set of deviations. By this method it would also be possible to weight the error of the snake robot head the most. Holden [13] has formulated an optimization problem that finds input torques by minimizing energy consumption while achieving propulsion along the desired path. The method is a great inspiration, but not quite yet a solution to the problem as it still requires the desired joint angles at the obstacles to be known.

6.3 Further insights from experiments

On the more constructive side, it is clear that the simulator has proven to be a great resource for presenting concepts and study the possibilities within obstacle aided locomotion. Furthermore, the modular architecture of the program, where controller, dynamics, path following etc. is decoupled, allows for it to be effortlessly modified.

The experiments have shown that the positioning of the obstacles and path in relation to each other is vital for the propulsion of the robot. This is especially the case in an environment where the possibility to aid friction for propulsion is absent. Furthermore, it has been shown that it is necessary to have a sufficient number of obstacles. Not only for the propulsion, but also for continuously aiding the robot with alignment along the path. Consequently, a limitation of the implemented system is that the configuration of obstacles and desired path need to be determined manually.

A further observation from the experiments, in particular 5.3.4, is that the obstacles used for aligning the rear part of the robot are comparable to a manipulator base. More specifically, the robot is in this position able to keep its rear links fixed in the perpendicular direction by pushing against these obstacles. Thus, the control of the proceeding links is increased in the perpendicular direction. This does in turn lead to the robot being able to get further with a greater number of links, as the robot slides through and stays in touch with the aligning obstacles for a longer period of time.

When it comes to the computational performance of the simulator, it is observed that the program requires significantly more time for computing the initialization of robots with 6 links or more than that of robots with fewer links.

An improvement would be defining the equations of motion directly rather than performing symbolic math differentiation to derive them. However, the initialization only has to run once for every configuration, and the real time visual performance of the actual simulation is still satisfactory for sample times greater than 0.1 ms.

Chapter 7

Conclusion

The project report has presented a MATLAB simulator developed to visualize the concepts of HPFC, OAL and the combination of the two, also known as HOAL. It has been a valuable resource for gaining more insight into the theory and existing ideas, as well as shed light on what challenges remain to be solved.

From the experiments, OAL is concluded to be a simple and intuitive concept with a lot of potential. This conclusion is drawn based on the experiments of the snake robot propelling forward. Despite the approximations made, the fraction of the HPFC idea applied to the simulator has proven that the inclusion of HPFC in realising the concept of HOAL is a step in the right direction. This is however not based on any direct use of HPFC, but rather the lack of it. More particularly, the fraction applied cannot be considered hybrid *position/force* control, as it is more of a shape control adapted to the constraints through the projections from the HPFC theory of West and Asada [4]. Trying to foresee the consecutive forces resulting from the movement and shape of the robot and thereby finding an optimal configuration of path and obstacles is a cumbersome operation. It is also a poor and very simplified way of controlling the propulsion, and thus it is obvious that explicit control of the forces should be included into this operation and that HPFC is a significant idea.

Consequently, it has been proven that the constraints, which are the foundation of the spaces in which shape and force can be controlled, are a significant part of the problem. In this project, the model of the robot dynamics excluded these constraint and instead introduced them in an approximation of the velocities, which again led to the laws of conservation of energy and momentum being violated. It is assumed that a more integrated and implicit definition of these constraints would enable the OAL method to benefit further from HPFC.

Finally, it can be stated that the concept of HOAL has been verified by means of analyzing visual experiments, but needs to be researched further for a more mathematical and solid basis. Future work regarding the challenges around this is presented below.

7.1 Future work

The most significant challenge per now is concluded to be defining and analyzing the constraints and resulting constraint forces between the snake robot and the environment. As the method of West and Asada [4] is not considering the dynamics of the system, it is proposed to adapt the method of Yoshikawa [5] to the snake robot in future work. This would yield a dynamical model with the obstacle constraints integrated, rather than introducing them explicitly and adding the resulting forces subsequently.

When it comes to the path following performance, a clear deficiency in the applied method is the calculation of the desired angles. As mentioned in Chapter 6, solving this challenge by means of an optimization problem would most probably return more satisfying results.

All experiments and calculations are based on the knowledge of the desired path. The path and obstacles are thus manually designed to achieve the presented results. However, a person will not necessarily see the most optimal path in a cluttered environment, nor is it a realistic action to define it manually in real life scenarios. Methods like model predictive control (MPC) or reinforcement learning (RL) are proposed for future work in finding the optimal path as the robot moves through the environment.

Bibliography

[1] Ø. Stavdahl. "Working note: Hybrid Position/Force Control for Perception Driven Obstacle Aided Locomotion (HOAL) in Snake Robots". unpublished. 2019.

[2] A. A. Transeth, R. I. Leine, C. Glocker, K. Y. Pettersen, and P. Liljebäck. "Snake robot obstacle-aided locomotion: Modeling, simulations, and experiments". In: *IEEE Transactions on Robotics* 24.1 (2008), pp. 88–104.

[3] M. H. Raibert, J. J. Craig, et al. "Hybrid position/force control of manipulators". In: *Journal of Dynamic Systems, Measurement, and Control* 103.2 (1981), pp. 126–133.

[4] H. West and H. Asada. "A method for the design of hybrid position/force controllers for manipulators constrained by contact with the environment". In: *Proceedings. 1985 IEEE International Conference on Robotics and Automation*. Vol. 2. IEEE. 1985, pp. 251–259.

[5] T. Yoshikawa. "Dynamic hybrid position/force control of robot manipulators – description of hand constraints and calculation of joint driving force". In: *IEEE Journal on Robotics and Automation* 3.5 (1987), pp. 386–392.

[6] T. Klafstad. “Hybrid Position/Force Control for Obstacle Aided Locomotion in Snake Robots”. unpublished. 2019.

[7] K. M. Lynch and F. C. Park. *Modern Robotics*. Cambridge University Press, 2017, pp. 272–286.

[8] K. M. Lynch and F. C. Park. *Modern Robotics*. Cambridge University Press, 2017, pp. 421–429.

[9] K. J. Waldron and J. Schmiedeler. “Kinematics”. In: *Springer Handbook of Robotics*. Springer, 2016, pp. 11–36.

[10] P. Liljebäck, K. Y. Pettersen, Ø. Stavdahl, and J. T. Gravdahl. *Snake robots: modelling, mechatronics, and control*. Springer Science & Business Media, 2012.

[11] R. M. Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017, pp. 155–189.

[12] E. Rezapour, K. Y. Pettersen, P. Liljebäck, J. T. Gravdahl, and E. Kelasidi. “Path following control of planar snake robots using virtual holonomic constraints: theory and experiments”. In: *Robotics and biomimetics* 1.1 (2014), p. 3.

[13] C. Holden, Ø. Stavdahl, and J. T. Gravdahl. “Optimal dynamic force mapping for obstacle-aided locomotion in 2d snake robots”. In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2014, pp. 321–328.

[14] Z. Y. Bayraktaroglu and P. Blazevic. “Understanding snakelike locomotion through a novel push-point approach”. In: (2004).

[15] S. Chiaverini, G. Oriolo, and I. D. Walker. “Kinematically redundant manipulators”. In: *Springer handbook of robotics* (2008), pp. 245–268.

[16] E. S. Conkur and R. Gurbuz. “Path planning algorithm for snake-like robots”. In: *Information Technology And Control* 37.2 (2008).

[17] I. The MathWorks. *Symbolic Math Toolbox*. Natick, Massachusetts, United State, 2019. URL: https://www.mathworks.com/help/symbolic/.