NTNU
Norwegian University of
Science and Technology

TECHNISCHE UNIVERSITÄT
ILMENAU

Technische Universität Ilmenau

Fakultät Informatik und Automatisierung
Fachgebiet Regelungstechnik

Masterarbeit

Hybrid Position and Force Control (HPFC) Framework for a Snake Robot Simulator

vorgelegt von

Justus Jentsch

Matrikelnummer:	61666
Studiengang:	Technische Kybernetik und Systemtheorie
verantw. Hochschullehrer:	Prof. Dr.-Ing. Johann Reger
wissenschaftlicher Betreuer:	Prof. Dr.-Ing. Øyvind Stavdahl
eingereicht am:	20.07.2025

Eigenständigkeitserklärung

Die vorliegende Arbeit habe ich selbstständig ohne Benutzung anderer als der angegebenen Quellen angefertigt. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Quellen entnommen wurden, sind als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer oder anderer Prüfungen noch nicht vorgelegt worden.

Ilmenau, 20.07.2025 Justus Jentsch

Acknowledgements

I would like to express my sincere gratitude to my supervisor at TU Ilmenau, Prof. Dr.-Ing. Johann Reger, for his valuable guidance and mentorship throughout the course of this thesis and my entire study program. His expertise and encouragement have been instrumental in shaping the direction and quality of my work.

I am also grateful to my supervisors at NTNU, namely, Prof. Dr.-Ing. Jan Tommy Gravdahl, for introducing me to this interesting research topic and for his warm and welcoming manner. I would like to extend my thanks to Prof. Øyvind Stavdahl and Irja Gravdahl, who supported my work in close collaboration. It was exciting to contribute to their ongoing research. I always felt welcome to ask questions or meet with them whenever time allowed. Their insightful feedback significantly shaped the content of this thesis.

I am truly thankful to have spent this final chapter of my studies in Trondheim. The people I have met here helped me grow as a person in ways that go far beyond academic development.

My deepest gratitude goes to Emelie, who joined me on this journey to Norway. Her unwavering support, patience, and care made it possible for me to take this step. More than anyone else, she understands the challenges I faced and has been my constant source of strength.

I would also like to thank my friends in Germany, especially my best friends Alexander and Till, which always had an open ear for my problems. Without their support, I would not have come this far.

But none of this would have been possible without my parents. Their constant support, patience, and strong belief in me have been the foundation for everything I have achieved. From early on, they encouraged me to be curious, resilient, kind, and to follow my dreams wholeheartedly. I owe this last thanks to them.

Abstract

Snake inspired robots offer promising capabilities for traversing complex and unstructured environments. Among various locomotion strategies, Obstacle-Aided Locomotion (OAL) presents a biologically inspired approach in which propulsion is achieved through interactions with external obstacles. Although the concept has been validated, practical implementations of OAL remain challenging due to the lack of reliable control frameworks capable of handling dynamic contact interactions.

This thesis investigates the application of Hybrid Position and Force Control (HPFC) to snake robots in the context of OAL. Building on the Hybrid Obstacle-Aided Locomotion (HOAL) concept recently proposed in the literature, the objective is to implement and evaluate an HPFC-based control strategy within a simulation environment.

A comprehensive modeling approach for a planar snake robot is developed, and two HPFC strategies are formulated: a dynamic variant and a passivity-based kinematic variant. One strategy is further integrated into a customized version of the SimSerpent simulation framework, which has been extended to support trajectory generation, obstacle placement, and precise system initialization and evaluation. This strategy is evaluated through multiple simulation scenarios, each designed to highlight specific challenges of HOAL. The results demonstrate the feasibility of HPFC for structured OAL scenarios and provide insights into the respective strengths and limitations of this approach.

The thesis concludes with a comprehensive discussion of the implementation challenges and outlines perspectives for future research on more complex trajectories for this approach.

Kurzfassung

Schlangenartige Roboter bieten vielversprechende Möglichkeiten zur Fortbewegung in komplexen und unstrukturierten Umgebungen. Unter den verschiedenen Fortbewegungsstrategien stellt die hindernisgestützte Fortbewegung (Obstacle-Aided Locomotion, OAL) einen biologisch inspirierten Ansatz dar, bei dem der Vortrieb durch Interaktionen mit externen Hindernissen erfolgt. Obwohl dieses Konzept bereits grundlegend validiert wurde, bleibt die praktische Umsetzung herausfordernd, da bislang zuverlässige Steuerungskonzepte zur Bewältigung dynamischer Kontaktinteraktionen fehlen.

Diese Arbeit untersucht den Einsatz der hybriden Positions- und Kraftregelung (Hybrid Position and Force Control, HPFC) bei Schlangenrobotern im Kontext von OAL. Aufbauend auf dem kürzlich eingeführten Konzept der hybriden hindernisgestützten Fortbewegung (Hybrid Obstacle-Aided Locomotion, HOAL) wird eine auf HPFC basierende Regelungsstrategie in einer erweiterten Simulationsumgebung implementiert und anschließend evaluiert.

Zu diesem Zweck wird ein umfassendes Modell eines planaren Schlangenroboters entwickelt und zwei HPFC-Strategien formuliert: eine dynamische Variante sowie eine passivitätsbasierte kinematische Variante. Eine dieser Strategien wird in eine angepasste Version der SimSerpent-Simulationsumgebung integriert, die um Funktionen zur Trajektorienerzeugung, Hindernisplatzierung sowie zur präzisen Initialisierung und Bewertung erweitert wurde. Die integrierte Strategie wird anhand mehrerer Simulationsszenarien evaluiert, die zentrale Herausforderungen der HOAL veranschaulichen. Die Ergebnisse belegen die prinzi-pielle Umsetzbarkeit von HPFC für strukturierte OAL-Szenarien und liefern Einblicke in die jeweiligen Stärken und Grenzen dieses Ansatzes.

Die Arbeit schließt mit einer Diskussion der Implementierungsherausforderungen und skizziert Perspektiven für zukünftige Forschung zu komplexeren Trajektorien.

Contents

List of Abbreviations

NTNU	Norwegian University of Science and Technology
OAL	Obstacle-Aided Locomotion
HPFC	Hybrid Position and Force Control
DHPFC	Dynamic Hybrid Position and Force Control
HOAL	Hybrid Obstacle-Aided Locomotion
DOF	Degree(s) Of Freedom
ROS	Robot Operating System
CAN	Controller Area Network
EE	End-Effector
MuJoCo	Multi-Joint Dynamics with Contact
e.g.	exempli gratia (engl.: for example)
i.e.	id est (engl.: that is)

List of Symbols

$\boldsymbol{q}_\mathrm{s}$	State vector containing joint angles and initial tail position and orientation
$\boldsymbol{q}_\mathrm{e}$	Extended state vector including environmental constraints
$\boldsymbol{\phi}$	Vector of joint angles
$\dot{\boldsymbol{\phi}}$	Vector of joint velocities
$\ddot{\boldsymbol{\phi}}$	Vector of joint accelerations
ϕ_i	Angle of the i-th joint
$\phi_{0,i}$	Initial angle of i-th joint
n	Total number of joints
θ_i	Angle of the i-th link
$\boldsymbol{\theta}$	Vector of link angles
l_i	Length of i-th link
m	Length of a link (for a snake robot with links of equal length)
M	Mass of a link (for a snake robot with links of equal mass)
I_i	Mass moment of inertia of i-th link
$\boldsymbol{r}_i$	Vector to center of mass of i-th link
$\boldsymbol{u}$	Vector of generalized velocities of the links
$\boldsymbol{v}_i$	Velocity of center of mass of i-th link
$\boldsymbol{\omega}_i$	Angular velocity of i-th joint
$\boldsymbol{Q}$	Generalized force acting along generalized coordinates $\boldsymbol{q}$
$\boldsymbol{J}(\boldsymbol{q})$	General robotic Jacobian
$\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})$	Lagrangian
$P(\boldsymbol{q})$	Potential Energy
$K(\boldsymbol{q}, \dot{\boldsymbol{q}})$	Kinetic Energy
$\boldsymbol{M}(\boldsymbol{q})$	Mass/inertia matrix of a robotic (snake) system
$d\mathbf{R}$	Measure accounting for normal contact impulses/forces, Coulomb friction impulses/forces, and joint constraint impulses/forces

g_{H}	Gap function, determining shortest distance between a snake robot and an obstacle
λ_{H}	Normal contact force magnitude
γ_{H}	Relative normal velocity between a link and an obstacle
$\boldsymbol{w}_{\mathrm{H}}$	Jacobian of the gap function g_{H}
γ_{T}	Tangential relative velocity
$\boldsymbol{\lambda}_{\mathrm{T}}$	Friction force vector
$\mathcal{C}_{\mathrm{T}}$	Convex set of admissible friction forces
μ_{T}	Coefficient of isotropic Coulomb friction
λ_{N}	Normal contact force magnitude (used in friction law)
$\mathbf{n}_i$	Contact normal vectors
$\boldsymbol{F}_{\mathrm{c}}$	General contact force
$\boldsymbol{F}$	Task space force vector
$r_{\mathrm{H}j}$	Position of the j-th obstacle
L_{H} / r_{C}	Radius of an obstacle
$r_{\mathrm{S}i}$	Contact point on the i-th snake robot link
L_{SC}	Radius of a semicircle forming part of a planar snake link body
ε_{C}	Width of the snake robot
$\mathcal{F}^{\mathrm{initial}}$	Inertial frame
$\mathcal{F}_i^{\mathrm{c}}$	Contact frames
$\mathcal{F}_i^{\mathrm{t}}$	Task frames
$\boldsymbol{\tau}$	Vector of control torques
$\underline{\boldsymbol{\tau}}$	Scalable torque vector
$\boldsymbol{\tau}_{\mathrm{c}}$	Control torque vector applied to actuated joints
τ_i	Control torque at i-th joint
$\boldsymbol{\tau}_{\mathrm{P}}$	Torque contribution from the position controller
$\boldsymbol{\tau}_{\mathrm{F}}$	Torque contribution from the force controller
$\boldsymbol{\tau}_{\mathrm{F}}^*$	Constraint torque from a contact
K_{P}	Proportional gain
K_{D}	Derivative gain
$x_{\mathrm{d}}(t)$	Desired position trajectory
$\phi_{i,\mathrm{d}}$	Desired angle of the i-th joint
$\boldsymbol{\theta}_r(t)$	Reference trajectory (for joint angles)
ω_{t}	Temporal frequency (for gait generation)
$\Delta\phi$	Spatial phase difference (for gait generation)
A_θ	Motion amplitude (for gait generation)

k	Scaling factor
$V(x)$	Non negative storage function
$w(x, y)$	Supply rate
$\boldsymbol{J}_\mathrm{t}$	Task Jacobian
$\boldsymbol{J}_\mathrm{P}$	Jacobian related to position control
$\boldsymbol{J}_\mathrm{F}$	Jacobian related to force control
$\boldsymbol{S}_\mathrm{p}$	Position selection matrix
$\boldsymbol{S}_\mathrm{f}$	Force selection matrix
$\boldsymbol{u}_\mathrm{p}$	Position control input vector
$\boldsymbol{u}_\mathrm{f}$	Force control input vector
$\ddot{\boldsymbol{r}}_\mathrm{t}^\mathrm{d}$	Desired task acceleration vector
$\ddot{\boldsymbol{r}}_\mathrm{t,P}^\mathrm{d}$	Desired task acceleration vector in position-controlled subspace
$\boldsymbol{f}_\mathrm{F}^\mathrm{d}$	Desired contact force vector
$\boldsymbol{f}_\mathrm{F}$	Constraint force vector expressed in the force-controlled subspace
$\boldsymbol{E}_\mathrm{P}$	Block-matrix selecting tangential direction for motion at contacts
$\boldsymbol{E}_\mathrm{F}$	Block-matrix selecting normal force directions at contact
$\boldsymbol{E}$	Transformation matrix transforming from inertial to task frames
$\boldsymbol{F}$	Projection filter/matrix
$\boldsymbol{a}_\alpha$	Input vector to motion controller
$\boldsymbol{a}_\phi$	Input vector to force controller
k_i	Contact distance to the i-th link
$\phi_{\mathrm{c},i}$	Contact angle of i-th link
$\boldsymbol{J}_{\mathrm{c},i}$	Contact Jacobian of i-th link
$\boldsymbol{v}_{\mathrm{c},i}$	Contact velocity vector of i-th link
$\boldsymbol{v}_{\mathrm{t},i}$	Task velocity vector of the i-th link
n_c	Number of hypersurfaces that constrain a snake robot
α_p	Obstacle attack angle for circular trajectory generation
r_P	Circular trajectory main radius
$2n_\mathrm{P}$	Number of trajectory segments
α	Global curve parameter
$[x_\mathrm{O}, y_\mathrm{O}]$	Coordinate origin
$[x_\mathrm{vj}, y_\mathrm{vj}]$	Initial position of the first joint (tail joint spawning position)
$\mathcal{M}_i(q_0)$	First order free motion set
$\mathcal{M}_i^{(2)}(q_0)$	Second order free motion set

List of Figures

List of Tables

1 Introduction

This first chapter introduces the subject and motivation for this thesis and outlines the contents that are relevant to the concept of snake robotics. A brief overview of the aspired control strategy gives context for the goal of snake robot locomotion using obstacles for locomotion. The prior work on this topic motivated this work and is addressed in the next chapter.

Figure 1.1: Selection of snake robots (from left to right: *ACM III* [Hir93], *Kulko* [LPSG12b], and *Wheeko* [LPSG12a]).

1.1 Motivation

Robots in various appearances have been present since the early 20th century. Often associated with humanoid features, robots have evolved to fulfill tasks in many industries. In robotics theory, manipulators typically refer to robotic arms that execute specific tasks, such as assembly, and are generally modeled as stationary systems [SSVO09]. Current mobile robots still face significant mobility limitations, often restricted by specific environmental conditions. Wheeled and legged robots have made considerable advancements over the past century, yet remain specialized for particular environments. An ideal mobile robot could be imagined as one that cannot only maneuver through all environments but also manipulate and interact with its surroundings.

The need for versatile mobility across diverse terrains inspired researchers to explore biological analogies, particularly drawing inspiration from snakes. The legless reptiles utilize mainly special scale arrangements and external objects to navigate their territory. The domain of snakes extends across almost all biomes. Whether slithering on solid ground, through jungle trees, deserts, or aquatic environments, snakes have adapted successfully to a variety of conditions. This makes snakes one of the most versatile animals on our planet when it comes to locomotion [RLT+21]. A selection of such snakes can be seen in Figure 1.2.

(a) Gliding tree snake [Raf25].

(b) Desert snake [Arn14].

(c) Aquatic sea snake [Bue19].

Figure 1.2: Examples of real-world snakes in diverse environments.

Currently, snake robots alone cannot be used effectively in e.g. collapsed structures or burning houses, and victims of catastrophes still rely on humanitarian help. The ambition is that snake inspired robots can be superior to all other robots in precisely these areas. The structural similarity of snake robots to industrial manipulators, such as robotic arms, further motivates their development and enables them to perform tasks such as cable laying or maintenance in contaminated, hazardous, or otherwise inaccessible environments. Building upon this realization, Hirose et al. presented the first approach for a snake-like robot as early as 1993 [Hir93]. Since then many snake robots have been built. A selection of snake robots can be seen in Figure 1.1.

Some approach locomotion with what is called *undulatory locomotion*. This is a type of motion that is characterized by wave-like movement patterns in order to propel forward. Another feature inspired by snakes is the structure of scales that offer anisotropic ground friction properties. This anisotropic friction property results in the ability for directional control and propulsion in biological snakes [LPSG13a]. While some robotic implementations have attempted to mimic these frictional properties directly, many have resorted to external propulsion methods such as nozzles, propellers, or wheels to simplify locomotion instead [Pet17].

Since biological snakes do not have these extensions, another method was introduced by Transeth et al. in 2008 [TLG+08]. *Obstacles-Aided Locomotion*, or OAL for short, is an approach where movement is generated solely through interactions with obstacles, while generating the right torque in the joints, resulting in the body pushing forward. This theory is strongly inspired by biological snake movement and has been experimentally validated for snake robots [TLG+08, TFL13]. A working control concept was later proposed and future work was based on it, but to this date only few reliable control strategies have been found for this concept [LPS09].

The approach discussed in this thesis focuses on the specialized appliance of a control strategy for OAL. *Hybrid Position and Force Control*, or HPFC for short, is a concept that was developed by Railbert and Craig in 1981 and is designed for the independent control of positions and forces of an end-effector [RC81]. The combination of HPFC with OAL was proposed by Irja Gravdahl et al. in 2022 and is called *Hybrid Obstacle-Aided Locomotion*, or HOAL for short [GSK+22]. This thesis builds upon the theoretical HOAL framework and explores the concept inside of a simulation environment. A precise definition of the problem is given in the following section.

1.2 Problem Definition

The scientific contribution of this thesis lies in advancing Hybrid Position and Force Control methods for Obstacle-Aided Locomotion in snake robots. The lack of a practical method for evaluating such control strategies presents an opportunity to address the following central research question:

> *Can Hybrid Position and Force Control be used to reliably move a simulated snake robot through a predefined and known obstacle arrangement using Obstacle-Aided Locomotion while maintaining essential contacts?*

Although significant progress has been made in the field of snake robotics, current control strategies for OAL are still mainly limited to theoretical frameworks or simplified, non-physics based simulations [GSK+22, TLG+08, LPSG13a]. In particular, there remains a lack of results presenting the effectiveness of HOAL approaches and their capability to manage the complex dynamic interactions between the robot and its environment.

This gap forms the basis for the present work, which aims to design and validate a comprehensive and customizable HPFC simulation framework for obstacle-aided snake robots. An overview of the specific contributions made in this thesis is provided in the section below.

1.3 Thesis Contributions

To contextualize the topic within current research, Chapter 2 reviews recent developments in snake robotics and HPFC strategies. It outlines relevant approaches and provides an overview of the field's complexity. The chapter also briefly describes the state of the simulation environment prior to this thesis.

Chapter 3 presents the modeling assumptions and the derivation of a mathematical model for a planar snake robot. Two control strategies are formulated and discussed: a dynamic HPFC approach and a passivity-based kinematic HPFC variant.

Chapter 4 describes the integration of the proposed methods into a modified version of the *SimSerpent* simulation framework. This includes the initialization of the snake robot, trajectory generation, obstacle configuration, and the incorporation of the control strategies to work within the simulation.

Chapter 5 presents the results of the tests carried out in the simulation. The test cases are explained and illustrated through simulation excerpts and plots that offer insight into the system behavior under the chosen control strategy.

Chapter 6 offers a comprehensive analysis of the simulation results. In addition, the chapter discusses the implementation around more complex trajectory examples and gives direction for future research in this field. Lastly, a documentation of challenges during the creation process, as well as a summary of the current state of the project is given.

Finally, Chapter 7 summarizes the main findings and provides a concluding assessment of the work.

2 State of the Art

In this chapter the current state of advancements in snake robot design is discussed. Furthermore, existing simulation and control frameworks are reviewed. Lastly, the development of Hybrid Position and Force Control (HPFC) and strategies relevant to Obstacle-Aided Locomotion (OAL) are outlined and discussed.

2.1 Snake Robots

As motivated in Chapter 1, snake robots are biologically inspired robots designed to mimic the undulating motion of serpentine animals, making them adept at navigating complex terrains, confined spaces, and irregular surfaces. Their modular and versatile nature has led to a variety of designs tailored for different tasks and environments.

The first snake robot *ACM III* was introduced by Hirose et al. in 1972 and documented comprehensively in 1993 [Hir93]. This project, which can be seen in Figure 1.1, laid the foundation for subsequent designs, which improved and varied on actuation, control strategies, and environmental adaptability.

Significant contributions from the NTNU in the past two decades include:

- *Kulko*, featuring advanced control for navigation in confined spaces [LPSG12b], as seen in Figure 1.1.
- *Wheeko*, designed for adaptability on uneven surfaces using lateral undulation [LPSG12a, LPSG13b]. It can be seen in Figure 1.1.
- *Mamba*, one of many aquatic snake robots developed and investigated at NTNU. While aquatic applications are outside the scope of this work, *Mamba* is one of the figurehead research projects designed for underwater exploration [LSPG14].
- *Boa*, incorporating enhanced proprioceptive sensing and precise control mechanisms. Figure 2.1 shows a conceptual render of this robot [LGVS23, Løw23]. Further specifications of the robot are also given in Section 2.2.1.

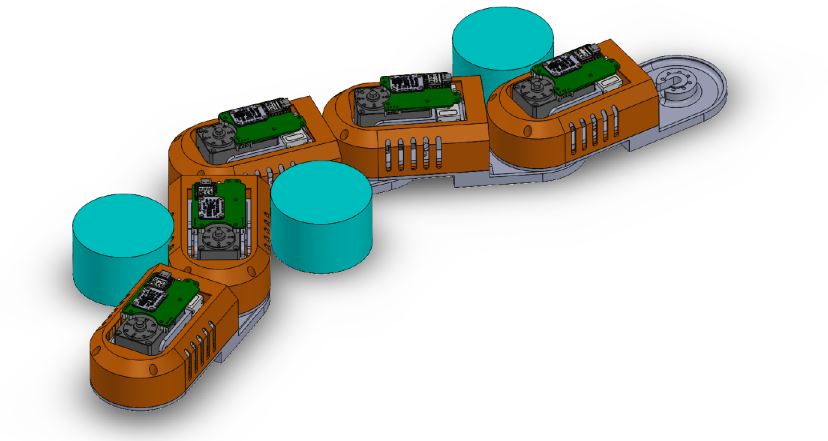


Figure 2.1: Render of a five-link prototype of the *Boa* snake robot [LGVS23].

There are numerous examples of snake-like robots from a variety of companies and countries. As many of these approaches focus on different locomotion strategies, they are not comparable to the ones named above. Liljebäck et al. [LPSG13a], among others, emphasize that new strategies for snake robots are commonly first tested in simulated environments. The following section discusses specifications to a selection of simulators used in NTNU projects.

2.2 Snake Robot Simulation at NTNU

The development of snake robot simulators is essential for testing locomotion strategies and control frameworks in realistic environments without physical prototypes. This saves time, resources, and allows users to directly see and influence control strategies while performing tests. In earlier work of Liljebäck or Transeth et al., simulations are carried out rather mathematically and are not explicitly based on physics engines [LPSG13a, LPSG12b, TFL13].

A selection of recent physics-based simulators are:

- *SnakeSIM*, a physics-based simulator developed within the snake robotics community, integrated with ROS and tailored for testing OAL [SSL18].
- *HOAL Simulator*, developed in a master thesis as an extension of *SnakeSIM*, incorporating basic functionality for evaluating Dynamic Hybrid Position and Force Control (DHPFC) in constrained environments [Kou20].
- *SimSerpent*, a modular and scalable simulator developed in a master thesis by Oscar Brunell Mørk, supporting detailed snake robot models and predefined testing scenarios but lacking trajectory support and advanced control strategies [Mør23].

The *Boa* project is the primary design inspiration [Løw23, Mør23] for *SimSerpent*. This simulator serves as the basis for this work and the framework is directly derived from the specifications of the *Boa*, which is explained in detail in the following section.

2.2.1 Hardware Platform: *Boa*

The platform under consideration is the *Boa* snake robot, as shown in Figure 2.1. This serpentine robot is the latest project from a group of researchers at the NTNU concerned with snake robots. Contributors are Øyvind Stavdahl, Jostein Løwer, and Irja Gravdahl among others. The *Boa* is designed for planar motion with obstacle interaction, and its links, which allow scalable lengths, provide the necessary degrees of freedom (DOF) to perform OAL in complex environments. Consequently, a model of a planar snake robot, as roughly depicted in Figure 2.2, is adequate for describing the dynamics of the *Boa*.

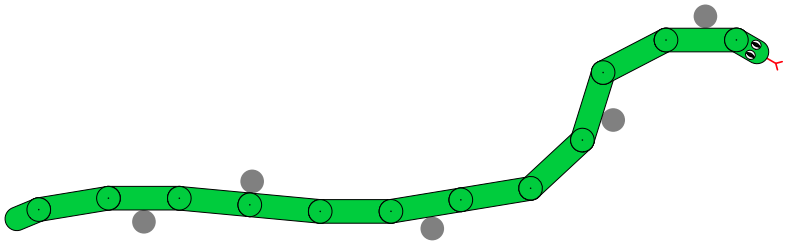


Figure 2.2: 13-link 2D snake model with obstacles resembling the *Boa* in operation.

This snake robot is still undergoing development, but Table 2.1 lists the specifications of the robot retrieved from an internal, unpublished documentation sheet [Sc25]. Consequently, the parameters can vary for future developments and are to be considered as estimates for the scope of this work. These specifications, some of which are later used in Chapter 4 for simulation purposes, provide details on the system structure, sensory technology, and actuation mechanisms employed in the prototype.

System Structure: The *Boa* snake robot is engineered as a modular system that integrates mechanical, electronic, and software components [Sc25]. The physical robot is set to comprise at least 13 interconnected links, with a dedicated head unit that serves as the primary communication interface with the control computer via an umbilical connection. This modular architecture allows for distributed sensing and actuation. Each link is equipped with its own servo and sensor modules and therefore the number of links is modular. Communication is managed over distinct buses: a servo bus (operating under Dynamixel Protocol 2.0) for precise joint actuation, and a separate CAN bus dedicated to sensor data [Dyn]. This separation not only ensures resilient, interference-free operation but also facilitates scalability and ease of maintenance.

Sensory Technology: The *Boa* platform incorporates advanced sensory technologies to enhance its interaction with dynamic environments [Sc25]. Integrated force sensors monitor the internal constraint forces between links, while an inertial measurement unit provides dynamic measurements, from which orientation can be estimated [ME-25, Ada14]. To ensure high fidelity data acquisition, the sensor modules communicate over a dedicated CAN bus. This approach minimizes interference with the servo network and guarantees reliable error handling and data integrity, thereby possibly enabling real-time obstacle detection and adaptive locomotion.

Actuation: Actuation in the *Boa* snake robot is achieved through high-performance Dynamixel servos [Dyn]. Distributed across the links, these actuators offer accurate and responsive joint actuation, which is suitable for executing complex locomotion patterns such as OAL. The servos operate under Dynamixel Protocol 2.0, which supports synchronized and accurate control. Moreover, the design allows for possible future enhancements, such as the integration of compliant actuators and flexible tendon mechanisms, to further emulate the organic movements of natural snakes and potentially improve energy efficiency in dynamic settings.

Table 2.1: Specifications of the *Boa* snake robot.

Parameter	**Specification**
Number of Links (DOF)	variable
Link Length	209.500 [mm]
Total Length	1769.500 [mm]
Link Weight	1747.314 [g]
Total Weight	22.715 [kg]
Actuators	Dynamixel XH540-V150R servos
Sensor Types	Force Sensor: K3D40±50N, IMU: Adafruit BNO055
Sensor Communication Protocol	CAN bus
Actuator Communication	Dynamixel Protocol 2.0 over RS485

Integrating this information into *SimSerpent* sets the foundation for a realistic simulation. *SimSerpent*'s specifications are explained in the following section.

2.2.2 Simulation Platform: *SimSerpent*

The simulator underlying this work is based on the master thesis of Oscar Brunell Mørk [Mør23] titled *SimSerpent: A Physics-based Simulator for a Next Generation Snake Robot*. Initially developed using *Isaac Sim*, the project eventually transitioned to a *MuJoCo*-based framework in order to leverage an efficient physics engine and rendering capabilities with *Python* connectivity [TET12].
SimSerpent provides a realistic and customizable simulation environment for snake robot dynamics, featuring detailed collision handling and the ability to reconfigure the snake robot and environment structure. Although the current version offers manual steering of the snake body without an integrated active controller, it supports extensive experimentation capabilities.

Overall, *SimSerpent* provides the groundwork for further integration of state estimation and control strategies, making it a valuable tool for advancing research on the *Boa* snake robot. The control approaches researched in the context of snake robots are discussed in the following section.

2.2.3 Control Approaches

Transeth and Pettersen's 2006 review [TP06] emphasizes that effective control of snake robots must address both the high degree of redundancy and the dynamic interactions with the environment. In later work, their framework was extended, considering simple feedback controllers, such as proportional (P), proportional-derivative (PD), and proportional-integral-derivative (PID) controllers [TLG+08]. These are applied at the joint level to regulate the motion of individual links. However, due to the complex coupling between joints, internal dynamics and the inherent nonlinearity of the system, these basic controllers are often insufficient for achieving reliable locomotion on uneven or unpredictable terrain.

To overcome these challenges, Transeth suggests that a dual control strategy is needed, one that combines position control with force feedback [TLG+08]. HPFC schemes are later proposed to integrate sensor measurements (e.g. contact forces) into the control loop, thereby allowing the snake robot to adapt its gait in real time during obstacle interactions [GSK+22]. This approach can be further extended with adaptive control algorithms to regulate both force and position, therefore compensating for model uncertainties and external disturbances.

Recent research by Patel and Dwivedy explores the control of three dimensionally modeled underwater snake robots [PD25]. They discuss snake robot dynamics and control, but its focus on 3D and underwater environments is outside the scope of this work.

Although *SimSerpent* [Mør23] currently does not feature any HPFC strategies, recent research by Irja Gravdahl et al. [GSK+22] highlights the potential benefits of incorporating such strategies. This thesis aims to explore this endeavor further. The objective is to balance computational efficiency with resilient performance in real-world scenarios.

This can be achieved by coordinating local joint controllers with a higher-level strategy that adapts the overall gait in response to environmental feedback. The functionality and potential benefits of HPFC are discussed further in Chapter 3. In the following Section 2.3, attention is turned to OAL, one of the most prominent approaches in modern snake robot locomotion.

2.3 Obstacle-Aided Locomotion (OAL)

Obstacle-Aided Locomotion is an established approach to snake locomotion and leverages the interaction between a snake robot and external obstacles to generate propulsion. This method, first explored by Transeth et al. [TLG+08], introduces control strategies and simulation results for planar surfaces. Later work of Transeth et al. also extends these strategies to sloped terrains [TFL13].

The concept presented by Transeth et al. remains state-of-the-art to this date and is therefore presented in detail. The most notable developments building upon the approach include a control and modeling framework [LPSG10, Lil11], a mathematical description of OAL dynamics [LGVS22], and recent advances in modeling and simulation techniques [GSK+22]. To motivate the concept of OAL, a brief summary of relevant definitions and notations is given in the following section.

2.3.1 Mathematical Framework for OAL

The framework for OAL is based on a non-smooth dynamics approach, which allows for the modeling of instantaneous changes in velocities caused by impacts with obstacles. The following equations and derivations are inspired by the 2008 work of Transeth et al. [TLG+08]. Their approach is based on the equality of measures:

$$\boldsymbol{M}d\boldsymbol{u} - d\mathbf{R} = \boldsymbol{\tau}_{\mathrm{c}}dt, \tag{2.1}$$

where:

$\boldsymbol{M}$	is the mass matrix of the snake robot,
$\boldsymbol{u}$	is the velocity vector of all links,
$d\mathbf{R}$	accounts for normal contact forces, Coulomb friction, and joint constraint forces,
$\boldsymbol{\tau}_\mathrm{c}$	is the control torque vector applied to the joints.

This unconventional form represents instantaneous changes in velocities to account for collisions. Therefore, the normal contact, friction, and constraint forces might not always be defined due to resulting infinite accelerations. In this case, impulses are used instead of forces.

Contact and Friction Forces: The normal contact force between the snake robot and an obstacle is governed by Signorini's law:

$$g_\mathrm{H} \geq 0, \quad \lambda_\mathrm{H} \geq 0, \quad g_\mathrm{H}\lambda_\mathrm{H} = 0,$$

where g_H is a gap function that determines the shortest distance between the snake robot and an obstacle, and λ_H is the normal contact force. This law ensures that the contact is impenetrable and can only transmit compressive forces. Coulomb friction at the contact point is expressed as

$$-\boldsymbol{\gamma}_\mathrm{T} \in \mathcal{N}_{\mathcal{C}_\mathrm{T}}(\boldsymbol{\lambda}_\mathrm{T}),$$

where $\boldsymbol{\gamma}_\mathrm{T}$ is the tangential relative velocity and $\boldsymbol{\lambda}_\mathrm{T}$ is the friction force. For isotropic Coulomb friction, the convex set of admissible friction forces $\mathcal{C}_\mathrm{T}$ is defined as

$$\mathcal{C}_\mathrm{T} = \{\boldsymbol{\lambda}_\mathrm{T} : \|\boldsymbol{\lambda}_\mathrm{T}\| \leq \mu_\mathrm{T}\lambda_\mathrm{N}\},$$

where μ_T is the friction coefficient and λ_N is the normal force.

Relative Velocity and Gap Functions: The relative velocity between a link and an obstacle is

$$\gamma_\mathrm{H} = \dot{g}_\mathrm{H} = \boldsymbol{w}_\mathrm{H}^\top \boldsymbol{u},$$

where $\boldsymbol{w}_\mathrm{H}$ is the Jacobian of the gap function g_H.

The gap function for the shortest distance between a link (modeled as a rectangle with semicircular ends) and a circular obstacle is:

$$g_{\mathrm{H}} = \|\boldsymbol{r}_{\mathrm{H}j} - \boldsymbol{r}_{\mathrm{S}i}\| - (L_{\mathrm{H}} + L_{\mathrm{SC}}),$$

where $\boldsymbol{r}_{\mathrm{H}j}$ is the position of the obstacle, $\boldsymbol{r}_{\mathrm{S}i}$ is the contact point on the snake robot link and L_{H} and L_{SC} are the radii of the obstacle and the semicircle, respectively. Building upon this framework, various control strategies have been explored. Relevant insights to this work are explained in the following Section 2.3.2.

2.3.2 Simple Control Strategy for OAL

To achieve simple OAL, the joint torques can be controlled using a proportional-derivative (PD) controller applied individually at each joint i:

$$\boldsymbol{\tau}_i = -K_{\mathrm{P}}(\phi_{i,\mathrm{d}} - \phi_i) - K_{\mathrm{D}}(\dot{\phi}_{i,\mathrm{d}} - \dot{\phi}_i), \quad \forall i \in \{1, \ldots, n\}, \tag{2.2}$$

where ϕ_i and $\phi_{i,\mathrm{d}}$ are the actual and desired joint angles, respectively. K_{P} and K_{D} are the proportional and derivative gains, and n is the total number of actuated joints. The desired joint angles $\phi_{i,\mathrm{d}}$ are typically based on requirements that result from a path. How a paths are created in the context of OAL and which properties they must comply with is explained in Chapter 3, Section 3.3.1. The goal is for the snake robot to generate sufficient propulsion while maintaining the desired shape and interaction with obstacles [TLG+08]. A general representation of a snake robot is depicted in Figure 2.3. This common illustration shows the referencing of the links, joints, torques and external forces.

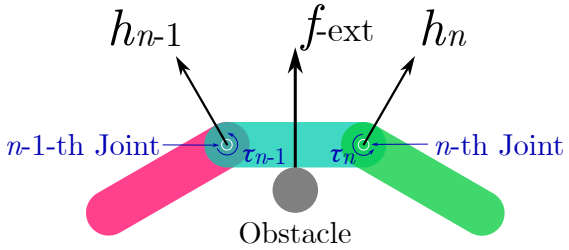


Figure 2.3: Simplified scheme of forces and torques acting on a 2-jointed planar snake in contact with one external object (inspired by [Mør23]).

OAL was explicitly developed for snake robots. Although HPFC has previously been suggested in literature as potentially beneficial, it has not yet been simulated within OAL contexts. HPFC itself was initially developed independently from snake robotics and thus represents a more general approach applicable across various robotic systems. The following section provides detailed context for the HPFC method, presenting its foundational concepts and recent developments relevant to this thesis.

2.4 Hybrid Position and Force Control (HPFC)

HPFC was first introduced by Raibert and Craig in 1981 [RC81] and further developed by West and Asada in 1985 [WA85]. This method addresses the control of robotic manipulators constrained by contact with the environment, enabling simultaneous control of position and force in separate orthogonal subspaces. The motion constraints define the directions in which contact forces can be actively controlled, and thus correspond to the *force-controlled subspace*. The *position-controlled subspace* describes the force constraints that restrict the development of contact forces and defines the directions in which unconstrained motion is possible. The resulting controller utilizes projection matrices to separate controllable directions. The equations of motion for the constrained manipulator are derived from the manipulator dynamics, evaluated at the points of contact. The control inputs are divided into position and force components with desired accelerations in the position and force subspaces.

The work of West and Asada [WA85] laid the groundwork for advanced HPFC controllers by systematically analyzing constraints and introducing projection-based controllers. This method has been successfully applied to a variety of constrained manipulator tasks, such as assembly, grinding, and obstacle interaction, providing a solid framework for simultaneous regulation of position and force.

Additionally, the historical development of control block diagrams for manipulators was thoroughly reviewed by Whitney in 1987 [Whi87], highlighting the evolution of HPFC frameworks. A critical analysis by Featherstone [FSK98] showed that Raibert and Craig's approach had limitations in certain edge cases. His work introduced adjustments to ensure the general applicability of the HPFC principle, thereby addressing scenarios that were previously unaccounted for.

Adaptive Position and Force Control: Recent research focused on enhancing HPFC through adaptive control methods. Tseng et al. [TJL24] explored adaptive position and force control using fuzzy neural networks, enabling real-time adjustment to nonlinearities and uncertainties in robotic systems. Additionally, Wang et al. [WAD+25] proposed an adaptive dynamic programming-based finite-time optimal backstepping control approach. This method ensures reliable and efficient force and position regulation for reconfigurable manipulators in complex and dynamic tasks.

Dynamic Hybrid Position and Force Control (DHPFC): Advancements in DHPFC were led by Yoshikawa [Yos87, YST88, YS93]. His research extended HPFC to dynamic systems, emphasizing the interplay between system inertia, constraints, and environmental interactions. Their framework forms the basis for modern implementations in scenarios requiring real-time responsiveness to dynamic changes. This could correspond to a grinding task in a real-world scenario, a concept that has been explored for snake-like robots performing similar constrained tasks like cutting. This topic was addressed by Nansai, et al. [NEI16] using a virtual internal model.

Hybrid Obstacle Aided Locomotion

In the context of snake robots, Hybrid Obstacle Aided Locomotion, or HOAL for short, is one of the latest research topics. It aims to combine the principles of OAL with HPFC to manage interactions between snake robots and their environment. This approach is set to improve consistent behavior, propulsion efficiency, and adaptability in cluttered environments. The HOAL framework extends classical HPFC concepts, originally developed for rigid manipulators, to hyper-redundant snake robots by partitioning their configuration space into orthogonal subspaces for motion and force control [Kou20, GSK+22].

Simulation based controllers are critical because effective performance in a realistic simulation environment is typically a prerequisite for successful real-world application. Consequently, rigorous testing of controllers within simulations that closely resemble actual dynamics is essential. Effective control strategies ensure that the robot can maintain stability, adapt to environmental changes, and accurately follow desired trajectories, all of which are essential for validating simulation results against practical performance. Advanced control strategies, such as HPFC, aim to enhance the agility and resilience

of snake robots operating in real-world environments. However, several critical questions remain unanswered. Many current models rely on idealized assumptions about friction and contact forces that may not hold in real-world scenarios, and the scalability of these approaches to fully three-dimensional, dynamic environments is still not proven. Moreover, the integration of control strategies with simulation platforms often overlooks practical issues, such as sensor noise and actuator limitations. Addressing these limitations is essential to bridge the gap between simulation and real-world implementation.

In summary, despite considerable progress in snake robot design and simulation, significant challenges remain. Current simulators typically do not incorporate HPFC strategies, and practical implementations of HPFC, particularly within complex, obstacle-rich environments, remain largely unexplored. These limitations emphasize the need for simulation frameworks capable of effectively combining theoretical HPFC models and practical performance—precisely the gap this thesis aims to address. The model for HOAL is outlined in following chapter. Furthermore, the overall theoretical background knowledge that this work is based on, as well as the methods for answering the initial question of the thesis, are discussed in the subsequent Chapter 3.

3 Hybrid Obstacle-Aided Locomotion (HOAL)

In this chapter, the theory behind two approaches to Hybrid Obstacle-Aided Locomotion (HOAL) is explained in detail. The first approach focuses on the dynamic HPFC by Yoshikawa et al. [Yos87] and the second on the kinematic approach by West and Asada [WA85]. The advantages and disadvantages of both approaches are discussed within this chapter. To motivate the general approach to HPFC, a simple example is chosen and demonstrated in the following section.

3.1 Hybrid Position and Force Control (HPFC) Motivated by an Example

This section provides an intuitive and technical introduction to HPFC. The concept is presented through a simplified planar manipulator example that essentially resembles key characteristics of the targeted snake robot system. The robot depicted in Figure 3.1 is known as a 2R-manipulator. It consists of two planar links actuated by two rotational joints. The first joint, with joint angle ϕ_1, is located at the origin $[x_\mathrm{O}, y_\mathrm{O}] = [0, 0]$. The links have respective lengths l_1 and l_2. In this depiction, both are set to one for simplicity. A horizontal wall is present in the workspace, and the robot is tasked with applying a constant contact force of $F_\mathrm{c} = 0.5\,\mathrm{N}$ to the wall, while executing a back-and-forth motion. This may correspond to a grinding task in a real-world scenario. The links are modeled as thin rods with centers of mass located at half their lengths. Their masses M_1 and M_2 are set to one and each has a corresponding moment of inertia (I_1, I_2). The robot is initialized with angles $\phi_{0,1} = 0$ and $\phi_{0,2} = \frac{\pi}{2}$.

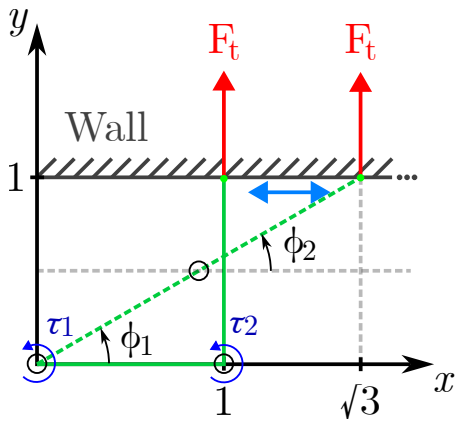


Figure 3.1: Planar robotic arm performing HPFC.

The positions of the centers of mass of the two links are given by

$$\boldsymbol{r}_1 = \begin{bmatrix} \frac{l_1}{2}\cos(\theta_1) \\ \frac{l_1}{2}\sin(\theta_1) \end{bmatrix}, \quad \boldsymbol{r}_2 = \begin{bmatrix} l_1\cos(\theta_1) + \frac{l_2}{2}\cos(\theta_1+\theta_2) \\ l_1\sin(\theta_1) + \frac{l_2}{2}\sin(\theta_1+\theta_2) \end{bmatrix},$$

where $\theta_1 := \phi_1 + \phi_{0,1}$ and $\theta_2 := \phi_2 + \phi_{0,2}$. The squared magnitudes of the velocities of the center of mass vectors are

$$|\boldsymbol{v}_1|^2 = \dot{\boldsymbol{r}}_1^\top \dot{\boldsymbol{r}}_1 = \frac{l_1^2}{4}\dot{\theta}_1^2,$$
$$|\boldsymbol{v}_2|^2 = \dot{\boldsymbol{r}}_2^\top \dot{\boldsymbol{r}}_2 = l_1^2\dot{\theta}_1^2 + \frac{l_2^2}{4}(\dot{\theta}_1+\dot{\theta}_2)^2 + l_1 l_2(\dot{\theta}_1+\dot{\theta}_2)\dot{\theta}_1\cos(\theta_2).$$

The angular velocities of the links are

$$\boldsymbol{\omega}_1 = [0\ 0\ 1]^\top \dot{\theta}_1, \quad \boldsymbol{\omega}_2 = \boldsymbol{\omega}_1 + [0\ 0\ 1]^\top \dot{\theta}_2.$$

Since gravity is neglected, the potential energy P vanishes. The kinetic energy

$$K(\boldsymbol{\theta}, \dot{\boldsymbol{\theta}}) = \frac{1}{2}a_1\dot{\theta}_1^2 + \frac{1}{2}a_3\cos(\theta_2)\dot{\theta}_1^2 + \frac{1}{2}a_2\dot{\theta}_2^2 + \frac{1}{2}\left(a_2 + a_3\cos(\theta_2)\right)\dot{\theta}_1\dot{\theta}_2,$$

with

$$\begin{aligned} a_1 &:= M_1\frac{l_1^2}{4} + M_2\frac{l_2^2}{4} + M_2 l_1^2 + I_1 + I_2, \\ a_2 &:= M_2\frac{l_2^2}{4} + I_2, \\ a_3 &:= M_2 l_1 l_2, \end{aligned}$$

is then used in the Euler-Lagrange formalism. Its partial derivatives with respect to the generalized coordinates $\boldsymbol{\theta}$ and their velocities are

$$\begin{aligned} \frac{d}{dt}\left(\frac{\partial K}{\partial \dot{\theta}_1}\right) &= \left(a_1 + a_3\cos(\theta_2)\right)\ddot{\theta}_1 + \frac{1}{2}\left(a_2 + a_3\cos(\theta_2)\right)\ddot{\theta}_2 - a_3\sin(\theta_2)\dot{\theta}_1\dot{\theta}_2 - \frac{1}{2}a_3\sin(\theta_2)\dot{\theta}_2^2, \\ \frac{\partial K}{\partial \theta_1} &= 0, \\ \frac{d}{dt}\left(\frac{\partial K}{\partial \dot{\theta}_2}\right) &= \frac{1}{2}\left(a_2 + a_3\cos(\theta_2)\right)\ddot{\theta}_1 + a_2\ddot{\theta}_2 - \frac{1}{2}a_3\sin(\theta_2)\dot{\theta}_1\dot{\theta}_2, \\ \frac{\partial K}{\partial \theta_2} &= -\frac{1}{2}a_3\sin(\theta_2)\dot{\theta}_1^2 - \frac{1}{2}a_3\sin(\theta_2)\dot{\theta}_1\dot{\theta}_2. \end{aligned}$$

Substituting into the Euler-Lagrange equations yields the dynamic model

$$\begin{aligned} \tau_1 &= \left(a_1 + a_3\cos(\theta_2)\right)\ddot{\theta}_1 + \frac{1}{2}\left(a_2 + a_3\cos(\theta_2)\right)\ddot{\theta}_2 - a_3\sin(\theta_2)\dot{\theta}_1\dot{\theta}_2 - \frac{1}{2}a_3\sin(\theta_2)\dot{\theta}_2^2, \\ \tau_2 &= \frac{1}{2}\left(a_2 + a_3\cos(\theta_2)\right)\ddot{\theta}_1 + a_2\ddot{\theta}_2 + \frac{1}{2}a_3\sin(\theta_2)\dot{\theta}_1^2. \end{aligned}$$

Rewriting this in terms of the configuration variables $\boldsymbol{\phi}$, the robot dynamics become

$$\begin{aligned} \underbrace{\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix}}_{=:\boldsymbol{\tau}} = &\underbrace{\begin{bmatrix} a_1 + a_3\cos(\phi_2+\phi_{0,2}) & \frac{1}{2}(a_2 + a_3\cos(\phi_2+\phi_{0,2})) \\ \frac{1}{2}(a_2 + a_3\cos(\phi_2+\phi_{0,2})) & a_2 \end{bmatrix}}_{=:\boldsymbol{M}(\boldsymbol{\phi})} \begin{bmatrix}\ddot{\phi}_1\\ \ddot{\phi}_2\end{bmatrix} \\ &+ \underbrace{\begin{bmatrix} -a_3\sin(\phi_2+\phi_{0,2})\dot{\phi}_2 & -\frac{1}{2}a_3\sin(\phi_2+\phi_{0,2})\dot{\phi}_2 \\ \frac{1}{2}a_3\sin(\phi_2+\phi_{0,2})\dot{\phi}_1 & 0 \end{bmatrix}}_{=:\boldsymbol{C}(\boldsymbol{\phi},\dot{\boldsymbol{\phi}})} \begin{bmatrix}\dot{\phi}_1\\ \dot{\phi}_2\end{bmatrix}. \end{aligned}$$

In compact form this can be written as

$$\boldsymbol{\tau} = \boldsymbol{M}(\boldsymbol{\phi})\ddot{\boldsymbol{\phi}} + \boldsymbol{C}(\boldsymbol{\phi},\dot{\boldsymbol{\phi}})\dot{\boldsymbol{\phi}}. \tag{3.1}$$

A traditional non-linear state-space model of the form

$$\dot{\boldsymbol{X}} = \boldsymbol{f}(\boldsymbol{X}) + \boldsymbol{g}(\boldsymbol{X})\boldsymbol{\tau} \tag{3.2}$$

can be derived, where the state vector is defined as

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{X}_1 & \boldsymbol{X}_2 \end{bmatrix}^\top = \begin{bmatrix} \phi_1 & \phi_2 & \dot{\phi}_1 & \dot{\phi}_2 \end{bmatrix}^\top.$$

The system dynamics can then be expressed as

$$\dot{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{X}_2 \\ -\boldsymbol{M}(\boldsymbol{X}_1)^{-1}\boldsymbol{C}(\boldsymbol{X}_1, \boldsymbol{X}_2)\boldsymbol{X}_2 \end{bmatrix} + \begin{bmatrix} \boldsymbol{0}_{2\times 1} \\ \boldsymbol{M}(\boldsymbol{X}_1)^{-1} \end{bmatrix} \boldsymbol{\tau}. \tag{3.3}$$

3.1.1 Application of HPFC for a 2-DOF Manipulator

With the kinematic and dynamic models of the planar so called 2R manipulator derived, the foundation is laid for the application of a HPFC strategy. The purpose of HPFC is to decouple the task space into two orthogonal subspaces: one in which the robot actively regulates its position, and another, in which it regulates the contact force exerted on the environment.

Task Specification: In the presented example, the robot end-effector is constrained to lie on a horizontal wall located at $y = 1$. The control objective is twofold:

1. Along the tangential x-direction: the end-effector is allowed to move freely and should track a desired position trajectory $x_\mathrm{d}(\cdot)$.
2. Along the normal y-direction: the end-effector should maintain a constant contact force $F_\mathrm{c} = 0.5\,\mathrm{N}$ directed into the wall.

This separation of goals motivates a dual control scheme that combines position and force control laws in orthogonal directions of the task space.

Mapping Joint Space to Task Space: To regulate forces and positions at the end-effector, the joint space control inputs $\boldsymbol{\tau}$ must be projected from task space quantities. The robot Jacobian $\boldsymbol{J}(\boldsymbol{\phi})$ maps joint velocities $\dot{\boldsymbol{\phi}}$ to end-effector velocities $\dot{\boldsymbol{x}}$ via

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{\phi})\dot{\boldsymbol{\phi}}.$$

Its transpose maps task space forces $\boldsymbol{F}$ to joint torques $\boldsymbol{\tau}$ through

$$\boldsymbol{\tau} = \boldsymbol{J}^\top(\boldsymbol{\phi})\boldsymbol{F}.$$

The Jacobian can be decomposed using selection matrices $\boldsymbol{S}_\mathrm{p}$ and $\boldsymbol{S}_\mathrm{f}$, which isolate the position-controlled and force-controlled directions, respectively:

$$\boldsymbol{J}_\mathrm{p} = \boldsymbol{S}_\mathrm{p}\boldsymbol{J}(\boldsymbol{\phi}), \qquad \boldsymbol{J}_\mathrm{f} = \boldsymbol{S}_\mathrm{f}\boldsymbol{J}(\boldsymbol{\phi}),$$

where in this example

$$\boldsymbol{S}_\mathrm{p} = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad \boldsymbol{S}_\mathrm{f} = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}.$$

Derivation of the End-Effector Jacobian: The Jacobian matrix $\boldsymbol{J}(\boldsymbol{\phi})$ encodes the linear relationship between joint velocity $\dot{\boldsymbol{\phi}}$ and the end-effector velocity $\dot{\boldsymbol{x}}$ in Cartesian space. For a planar 2R manipulator, the forward kinematics of the end-effector located at the end of the second link can be written as

$$\boldsymbol{x}(\boldsymbol{\phi}) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} l_1\cos(\phi_1 + \phi_{0,1}) + l_2\cos(\phi_1 + \phi_{0,1} + \phi_2 + \phi_{0,2}) \\ l_1\sin(\phi_1 + \phi_{0,1}) + l_2\sin(\phi_1 + \phi_{0,1} + \phi_2 + \phi_{0,2}) \end{bmatrix}.$$

Taking the total time derivative gives the Cartesian velocity of the end-effector

$$\dot{\boldsymbol{x}} = \frac{d}{dt}\boldsymbol{x}(\boldsymbol{\phi}) = \boldsymbol{J}(\boldsymbol{\phi})\dot{\boldsymbol{\phi}},$$

where the Jacobian $\boldsymbol{J}(\boldsymbol{\phi})$ is defined by the partial derivatives of the position with respect to the joint angles:

$$\begin{aligned} \boldsymbol{J}(\boldsymbol{\phi}) &= \begin{bmatrix} \dfrac{\partial x}{\partial \phi_1} & \dfrac{\partial x}{\partial \phi_2} \\ \dfrac{\partial y}{\partial \phi_1} & \dfrac{\partial y}{\partial \phi_2} \end{bmatrix} \\ &= \begin{bmatrix} -l_1\sin(\phi_1 + \phi_{0,1}) - l_2\sin(\phi_1 + \phi_{0,1} + \phi_2 + \phi_{0,2}) & -l_2\sin(\phi_1 + \phi_{0,1} + \phi_2 + \phi_{0,2}) \\ l_1\cos(\phi_1 + \phi_{0,1}) + l_2\cos(\phi_1 + \phi_{0,1} + \phi_2 + \phi_{0,2}) & l_2\cos(\phi_1 + \phi_{0,1} + \phi_2 + \phi_{0,2}) \end{bmatrix} \end{aligned}$$

Each column of the Jacobian corresponds to the contribution of the respective joint to the end-effector velocity.

Note that:

- The Jacobian depends only on the current joint angles.
- It is sufficient to derive this using symbolic differentiation of the forward kinematics.
- For force control, the transpose $\boldsymbol{J}^\top$ maps Cartesian forces to joint torques.

The Jacobian is used in the force and position control law to correctly map control commands from task space into joint space.

Position and Force Control Law: The control input $\boldsymbol{\tau}$ is then computed as a superposition of two components:

$$\boldsymbol{\tau} = \boldsymbol{J}_{\mathrm{p}}^\top \boldsymbol{u}_{\mathrm{p}} + \boldsymbol{J}_{\mathrm{f}}^\top \boldsymbol{u}_{\mathrm{f}},$$

where

$\boldsymbol{u}_{\mathrm{p}}$ is a conventional position control input (e.g. from a PD controller) operating in the unconstrained x-direction and

$\boldsymbol{u}_{\mathrm{f}}$ is a force control input that maintains the desired force in the constrained y-direction.

This formulation guarantees orthogonality between force and position control actions, assuming the constraint surface is perfectly rigid and known.

Feedback Structure: To implement the controller, measurements of both position and contact force at the end-effector are required. The end-effector position is computed via forward kinematics $\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{\phi})$, and the contact force can be estimated using force sensors or inferred from the dynamics via inverse statics. The control law is typically implemented in the joint space as part of a dynamic simulation loop:

$$\dot{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{X}_2 \\ \boldsymbol{M}(\boldsymbol{X}_1)^{-1} \left(\boldsymbol{\tau} - \boldsymbol{C}(\boldsymbol{X}_1, \boldsymbol{X}_2)\boldsymbol{X}_2 \right) \end{bmatrix},$$

where this representation can be derived from Equation (3.3). This closed-loop control structure allows the robot to regulate motion and contact forces simultaneously, enabling interaction with the environment in a safe and compliant manner.

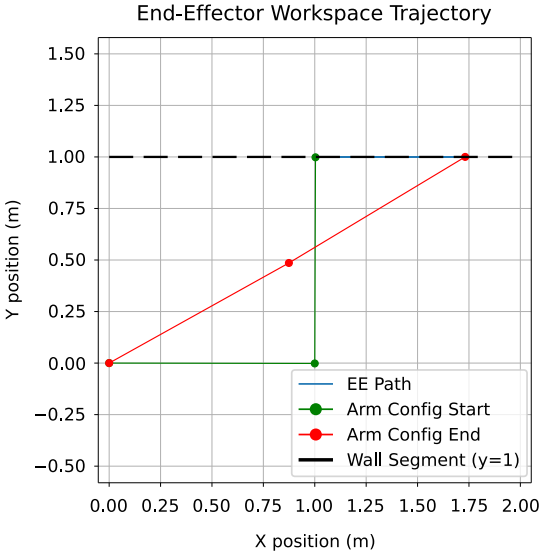


Figure 3.2: Start and end positions of a 2-DOF robotic manipulator.

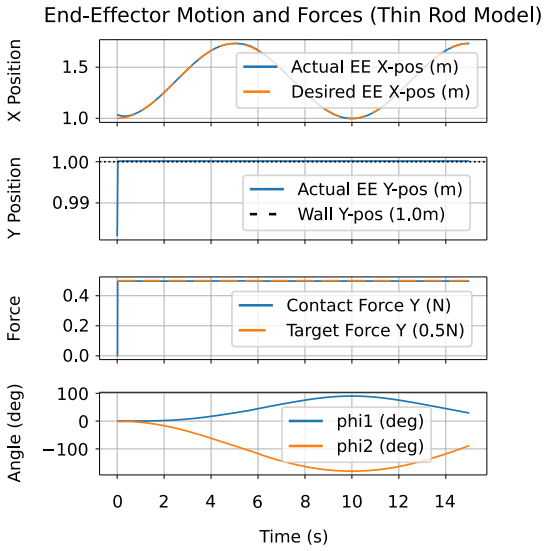


Figure 3.3: Time evaluation of a HPFC controller for a 2-DOF manipulator.

3.1.2 Simulation of a Planar Robotic Manipulator Controlled with HPFC

The simulation validates the effectiveness of the HPFC strategy when applied to the derived dynamic model. The outcome is visualized in Figures 3.3 and 3.2. Figure 3.2 shows the execution of the scenario illustrated previously in Figure 3.1. The end-effector (EE) completes one and a half back-and-forth motions along the constraint surface (Wall). As shown in the "X Position" plot of Figure 3.3, the motion starts with a small offset. The "Y Position" plot confirms that the EE maintains contact with the wall without penetrating it. This is reflected in the applied contact force, as shown in the "Force" plot. Finally, the "Angle" plot presents the joint angles over time, which remain smooth and free of oscillations or abrupt discontinuities.

Additional simulations were conducted under varied initial conditions and controller gains. The results indicate that HPFC performance is sensitive to both the initial pose and the tuning of the PD position controller. However, these results should not divert the focus away from the actual ambition. For this reason, no detailed evaluation is provided here. Nonetheless, it shows that for systems with well identified dynamics and carefully selected gains, HPFC proves to be a reliable and effective control strategy for constrained manipulation tasks.

This section serves to demonstrate the complexity involved in deriving a dynamic model, even for a simple 2-DOF manipulator. The modeling challenge grows significantly for higher-DOF systems, such as snake robots, which are typically modeled as floating-base manipulators. The following chapter builds upon this complex topic and discusses the different approaches of HPFC in snake robotics.

3.2 Modeling for Hybrid Position and Force Control (HPFC) in Snake Robotics

In general, modeling snake robots requires highly complex, non-linear dynamics due to their numerous degrees of freedom and intricate interactions with the environment. However, certain dynamic properties, such as out-of-plane movements or specific frictional effects, can be reasonably neglected in the planar scenarios considered in this thesis. The justification for these simplifications is provided throughout this section, beginning with an explanation of the unconstrained dynamics.

3.2.1 Unconstrained Dynamics

This section is mainly inspired by a conference paper of Irja Gravdahl et al. from 2022, concerning the theory of Dynamic Hybrid Position and Force Control (DHPFC) in OAL. The following equations are adopted from this article in a slightly modified form [GSK+22]. The robotic equations of motion for a planar snake robot are described as a hyper-redundant, floating-base manipulator:

$$\boldsymbol{M}(\boldsymbol{q}_\mathrm{s})\ddot{\boldsymbol{q}}_\mathrm{s} + \boldsymbol{h}(\boldsymbol{q}_\mathrm{s}, \dot{\boldsymbol{q}}_\mathrm{s}) = \boldsymbol{\tau}_\mathrm{s}, \tag{3.4}$$

where

$\boldsymbol{M}$ is the inertia matrix,

$\boldsymbol{h}$ includes Coriolis and centrifugal terms,

$\boldsymbol{\tau}_\mathrm{s} = \begin{bmatrix} \mathbf{0}_{1\times 3} & \boldsymbol{\tau}^\top \end{bmatrix}^\top$ is the control torque vector, and

$\boldsymbol{q}_\mathrm{s} = \begin{bmatrix} x_0 & y_0 & \phi_0 & \phi_1 & \dots & \phi_{n-1} \end{bmatrix}^\top$ is the state vector, consisting of the tail pose (x_0, y_0, ϕ_0) and joint angles ϕ_i.

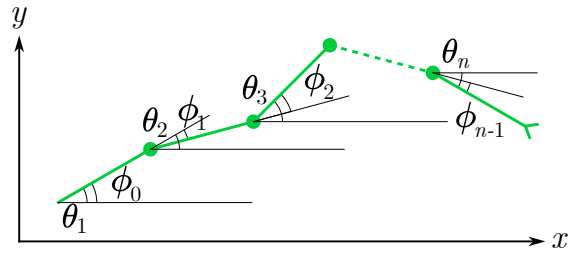


Figure 3.4: Introduction of local segment angles.

The joint angles ϕ_i are related to the link angles θ_i by

$$\phi_i = \theta_{i+1} - \theta_i. \tag{3.5}$$

These angles are shown in Figure 3.4. The following section explains how the state vector expands due to the restrictions imposed by obstacles.

3.2.2 Constrained Kinematics

When introducing environmental constraints, the extended state vector is

$$\boldsymbol{q}_\mathrm{e} = \begin{bmatrix} \boldsymbol{q}_\mathrm{s}^\top & k_1 & \dots & k_{n_\mathrm{c}} & \phi_{\mathrm{c},1} & \dots & \phi_{\mathrm{c},n_\mathrm{c}} \end{bmatrix}^\top, \tag{3.6}$$

where k_i and $\phi_{\mathrm{c},i}$ represent the contact distances and angles. The velocities at the contact points are related to the extended state vector by

$$\boldsymbol{v}_{\mathrm{c},i} = \boldsymbol{J}_{\mathrm{c},i}(\boldsymbol{q}_\mathrm{e})\dot{\boldsymbol{q}}_\mathrm{e}, \tag{3.7}$$

where $\boldsymbol{J}_{\mathrm{c},i}(\boldsymbol{q}_\mathrm{e})$ is the contact Jacobian. The constraint forces $\boldsymbol{f}_\mathrm{F}$ are expressed in the force-controlled subspace defined by

$$\boldsymbol{f}_\mathrm{F} = \boldsymbol{E}_\mathrm{F}^\top \boldsymbol{f}, \tag{3.8}$$

where $\boldsymbol{E}_\mathrm{F}$ is a section matrix for force constraints at each contact point.

3.2.3 Integrated HPFC Framework: Combining Dynamics and Kinematics

The general dynamic equations for a snake robot interacting with the environment under contact constraints are expressed as

$$\boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\ddot{\boldsymbol{q}}_\mathrm{e} + \boldsymbol{h}(\boldsymbol{q}_\mathrm{e}, \dot{\boldsymbol{q}}_\mathrm{e}) = \boldsymbol{\tau}_\mathrm{c} - \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{f}, \tag{3.9}$$

where $\boldsymbol{J}_\mathrm{t}$ is the task Jacobian. Within the HPFC framework, the control torque $\boldsymbol{\tau}_c$ can be decomposed into two distinct components:

$$\boldsymbol{\tau}_\mathrm{c} = \boldsymbol{\tau}_\mathrm{P} + \boldsymbol{\tau}_\mathrm{F}, \tag{3.10}$$

where
$\boldsymbol{\tau}_\mathrm{P}$ is the torque contribution from the position controller, and
$\boldsymbol{\tau}_\mathrm{F}$ is the torque contribution from the force controller.

These separate control contributions enable simultaneous regulation of position and contact forces. The HPFC controller components are

$$\boldsymbol{\tau}_\mathrm{P} = \boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\boldsymbol{a}_\alpha + \boldsymbol{h}(\boldsymbol{q}_\mathrm{e}, \dot{\boldsymbol{q}}_\mathrm{e}), \tag{3.11}$$

$$\boldsymbol{\tau}_\mathrm{F} = \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{E}_\mathrm{F}^\top \boldsymbol{a}_\phi, \tag{3.12}$$

where $\boldsymbol{a}_\alpha$ and $\boldsymbol{a}_\phi$ are the inputs to the motion and force controllers. The desired accelerations in the task space are determined through

$$\boldsymbol{a}_\alpha = \boldsymbol{J}_\mathrm{t}^\dagger \left(\ddot{\boldsymbol{r}}_\mathrm{t}^\mathrm{d} - \dot{J}_\mathrm{t}\dot{\boldsymbol{q}}_\mathrm{e} \right), \tag{3.13}$$

where $\boldsymbol{J}_\mathrm{t}^\dagger$ is the pseudo-inverse of the task Jacobian, and $\ddot{\boldsymbol{r}}_\mathrm{t}^\mathrm{d}$ is the desired task acceleration. The proposed HPFC strategy inherently involves constraints, not only imposed by the kinematics of the robot, but also by its interactions with the environment. Therefore, a clear definition and explicit treatment of these constraints is necessary. It is provided in the following Section 3.2.4.

3.2.4 Constraints on Motion and Force

Effective control of snake robots interacting with obstacles requires careful handling of both motion and force constraints. To achieve this, the configuration space is partitioned into motion ($\mathbb{M}$) and constraint ($\mathbb{C}$) subspaces with

$$\boldsymbol{E} = \begin{bmatrix} \boldsymbol{E}_{\mathrm{P}} \\ \boldsymbol{E}_{\mathrm{F}} \end{bmatrix}, \tag{3.14}$$

where $\boldsymbol{E}_{\mathrm{P}}$ is defined as a block matrix selecting the tangential direction for allowable motion at each contact point, while $\boldsymbol{E}_{\mathrm{F}}$ selects the normal directions at these contact points to ensure the force constraints. The section below provides a detailed explanation for the derivation of these spaces in the context of OAL.

3.3 Hybrid Obstacle-Aided Locomotion (HOAL) Framework

Building upon the general definitions of motion and force constraints, this section focuses explicitly on the combination of HPFC and OAL. This novel integration, referred to as Hybrid Obstacle-Aided Locomotion (HOAL), extends classical HPFC methods to hyper-redundant snake robots interacting dynamically with their environment [GSK+22]. To motivate this concept, a look at the configuration space $\mathbb{Q}$ is advised. For traditional robot manipulators, this space is divided into the motion space and the force space.

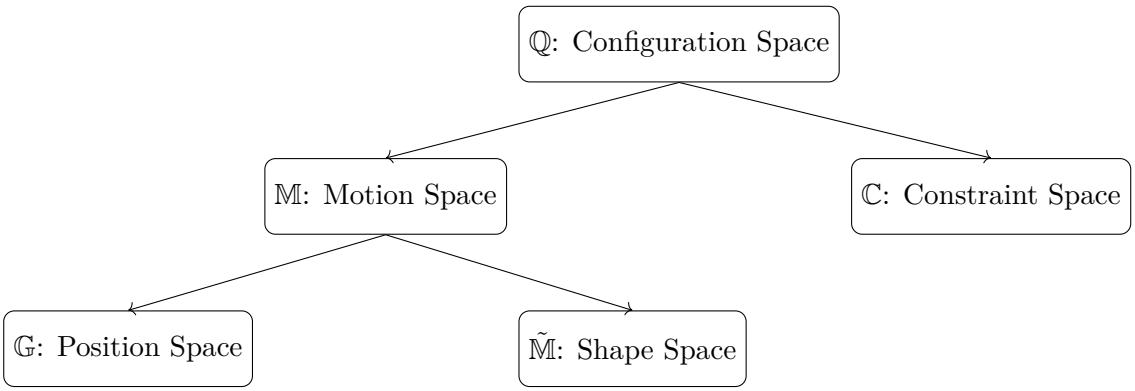


Figure 3.5: Division of the configuration space into motion and constraint spaces.

In Railbert and Craig [RC81], the tangent subspaces are orthogonal to each other. They refer to the spaces as the position controlled and force controlled subspace. In West and Asada [WA85], they are referred to as the allowable position and force space. In the context of this work, the position controlled subspace or the allowable position space, is referred to as the motion space $\mathbb{M}$. This space is again divided into the Position Space $\mathbb{G}$, referring to the global robot pose, and into the shape space $\tilde{\mathbb{M}}$, describing the relative arrangement of the snake robots parts. Lastly, the force controlled subspace or the allowable force space is denoted as the constraint space $\mathbb{C}$ [WA85]. This division is depicted in Figure 3.5, and the difference between traditional robot manipulators and snake robots in this context are outlined by Table 3.1.

	↙ $\mathbb{Q}$: Configuration Space ↘	
	Motion Space $\mathbb{M}$	Force (Constraint) Space $\mathbb{C}$
Traditional Robot Manipulators	DOFs are tangent to a contacting rigid surface; i.e., directions in which end-effector can still move when in contact.	DOFs are normal to a contacting rigid surface; i.e., directions in which end-effector motion is constrained and where force can be applied.
Snake Robots	Shape-change DOFs and directions are tangent to frictionless obstacles; i.e., relative joint movements and tangential slipping motions that produce locomotion.	DOFs are normal to each contacting obstacle; i.e., directions in which motion is blocked and forces are exchanged.

Table 3.1: Comparison of configuration spaces for traditional robot manipulators and snake robots.

An intuitive example for $\mathbb{C}$ can be constructed. Imagine a snake robot in the configuration of Figure 3.6. For this configuration, it is possible to find a three dimensional scalable torque vector $\underline{\boldsymbol{\tau}} = k \begin{bmatrix} \boldsymbol{\tau}_1 & \boldsymbol{\tau}_2 & \boldsymbol{\tau}_3 \end{bmatrix}^\top$, $k \in \mathbb{R}$ that does not induce any movement. The subspace that represents such torques is the constraint space $\mathbb{C}$. Vice versa, the motion space can be seen as the set of DOF in which movement can occur freely, meaning no torque or force along these directions will alter static contact forces. Movements that would be constrained by obstacles, such as pushing directly into them, are explicitly excluded from this space. This is further refined by introducing a predefined path within the position space $\mathbb{G}$, restricting motion directions accordingly. The shape space $\tilde{\mathbb{M}}$ is defined by torques that alter only the internal configuration of the snake, without changing its global position. Thus, the following distinctions hold: In the constraint space $\mathbb{C}$,

arbitrary forces can be applied without causing movement, whereas in the motion space $\mathbb{M}$, arbitrary movements can occur without causing any changes in static contact forces. In the scope of this work, dynamic effects due to acceleration are excluded. To guide these permitted movements, a path definition is required. The concept of a path in the context of snake robotics and HOAL is explained in the following section.

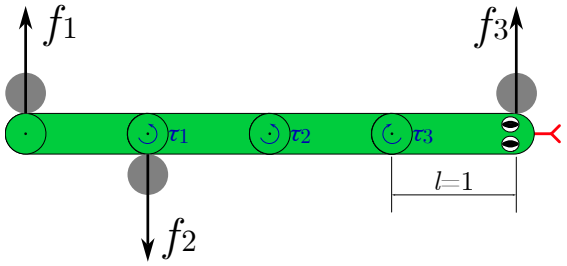


Figure 3.6: Visualization of a constrained snake robot.

3.3.1 Snake Robot Paths

A *path* in the context of HOAL is defined as a continuous curve that coincides with a specified start and end pose, and that the snake robot can traverse while every link center remains on the curve at all times. When the robot maintains form closure contacts along such a curve, only forward motion along the curve is kinematically permissible, ensuring that the robot cannot drift away from the prescribed route. The term form closure is formally defined in Section 3.3.2. The definitions of this section are stated in a reviewed but unpublished paper from Irja Gravdahl et al. [GLPS25]. Paths suitable for HOAL can vary depending on the scenario, however, a path should satisfy at least the following assumptions.

Assumptions on the Path: The path from a start to an end pose can be assumed to be known if the following conditions are met [GLPS25]:

1. It is a path with a form closure margin [RB98] and allowable radii for the robot to follow.
2. At $t = 0$, the snake robot is initialized on the path.
3. The corresponding references for both motion and force are known for every time step in the task-frames $\left(\ddot{\boldsymbol{r}}^{\mathrm{d}}_{\mathrm{t,P}}, \boldsymbol{f}^{\mathrm{d}}_{\mathrm{F}}\right)$.

This study assumes that a feasible path is available and satisfies the geometric constraints, such as allowable curvature and a non-zero form closure margin. The development of algorithms for constructing or optimizing such paths is considered beyond the scope of this work. These findings outline the path framework, but for a realistic snake robot model, additional assumptions have to be made. This is done in the following section.

3.3.2 Assumptions about the Environment and the Snake Body

To enable the framework of HOAL, certain assumptions must be made about the robot and its environment. These assumptions simplify the physical model while retaining essential behaviors relevant for testing control strategies.

1. The snake can move freely in each of its joints in the absence of external obstacles.
2. The joints themselves are only constrained by an allowable angular range. This range is derived from the real-world platform *Boa* [Løw23, Sc25].
3. The individual links that make up the snake are assumed to be rigid, and the joints connecting them are ideally actuated.
4. Obstacles are modeled as frictionless, rigid, and immovable points with known positions. Interactions are introduced by the constrained kinematics.
5. The surface on which the snake robot moves is frictionless and horizontal. If the snake makes contact with an obstacle, the body cannot be penetrated.
6. Dynamics can be ignored ($\ddot{\boldsymbol{x}} \equiv \mathbf{0}$) when movements are carried out very slowly. The snake is considered to be a passive system.

As shown later in Section 3.3.4, the last point is of high importance and forms the basis for a different approach to the problem. For the following Section 3.3.3, this assumption is initially excluded. The concept of form closure is crucial for justifying the assumption of predefined contact points and propulsion through environmental interactions. This concept is briefly explained below.

Form Closure: Concept and Relevance

The following definition of *form closure* is adopted from Rimon and Burdick's 1998 seminal work on grasp analysis [RB98]. Let B denote either a segment or the entire body of a (snake) robot at a given configuration q_0 in simultaneous contact with k environmental obstacles. For each contact i, let $\mathcal{M}_i(q_0) \subset T_{q_0}\mathbb{R}^m$ be the first order free motion set, that is the set of all allowable instantaneous velocity vectors at the contact that do not violate the geometric constraints imposed by the obstacle. Then, B is said to be in *first order form closure* if

$$\bigcap_{i=1}^{k} \mathcal{M}_i(q_0) = \{0\},$$

meaning that the only admissible instantaneous motion consistent with all the contact constraints is the trivial (zero) motion.

Similarly, B is in *second-order form closure* if an analogous condition holds for the second-order motion cones, which incorporate both velocity and acceleration constraints. Specifically, the intersection of the second-order free motion sets reduces to the zero vector:

$$\bigcap_{i=1}^{k} \mathcal{M}_i^{(2)}(q_0) = \{0\}.$$

This implies that, even when considering permissible accelerations consistent with the contact geometry, no nontrivial motion is allowed. A more detailed treatment of this condition is provided in [RB98].

Relevance to snake robot locomotion: In grasping, form closure implies that the object is fully immobilized by contacts. For a snake robot in an obstacle rich environment, one can similarly achieve form closure of the body or a body segment when enough contacts interlock to constrain all free motions. However, unlike a static grasp of an object, snake locomotion can, under appropriate conditions, maintain form closure continuously. This can be achieved by establishing new contacts with a suitably oriented normal force before losing an existing one. For example, if the robot ensures a given minimum number of contacts and contact transitions are timed so that a new contact enters a form closure-supporting orientation prior to the release of the aftmost contact,

continuous first-order form closure can be maintained throughout the motion. The continuous maintenance of form closure requires the following assumptions: the availability of suitably placed obstacles, sufficient contact points, and an appropriate path.

Some locomotion strategies deliberately break local form closure in parts of the body to reconfigure or advance other parts. In those cases, form closure is maintained only locally or momentarily and is then released and reestablished to effect progression. Whether one aims for continuous form closure or intermittent breaking depends on the environment geometry, friction assumptions, and path design.

Hybrid Dynamical System Properties

The term *Hybrid Dynamical System* is referenced from the work of Goebel et al. [GST12]. These systems are understood as ones which "combine behaviors that are typical of continuous-time dynamical systems with behaviors that are typical of discrete-time dynamical systems" [GST12].

In the context of snake robotics, this can be found in the continuous movement patterns and the instantaneous gain or loss of obstacle contacts. In Chapter 2, Section 2.3.1 this is indicated by Equation 2.1 in the context of OAL. Irja Gravdahl et al. [GSK+22] found that when a snake robot moves from a start to an end position, it transitions through four discrete events:

1. To gain contact with an obstacle.
2. To lose contact with an obstacle.
3. The contact point moves across a joint, now acting on a new link.
4. Using an obstacle in contact for propulsion.

This holds until the snake robot is in contact with multiple objects. The movement behavior of the snake is continuous. This also means that the dimensionality of the subspaces may change as the snake robot moves along its path. These discrete transitions (e.g., gaining or losing contact with obstacles) that occur alongside continuous motion define a hybrid dynamical system, as described by Goebel et al. [GST12].

When coming back to the last assumption of the environment and snake body, it is important to note that, concerning neglecting the dynamics (i.e., $\ddot{\boldsymbol{x}} \equiv \mathbf{0}$), a separate discussion of stability is redundant. Because the dynamics are neglected and the snake is modeled as a passive system with idealized, frictionless, and immovable obstacles, the system is considered to be in a quasi-static equilibrium. This is further explained in Section 3.3.4. In this context, the geometric and kinematic constraints, such as the form closure margin along the path, ensure inherent stability of the robot configuration. To support these transitions and to enable motion under changing contact conditions, the following Section 3.3.3 presents a control strategy based on dynamic modeling.

3.3.3 Dynamic Controller for HPFC

As introduced in Section 3.2.1, the snake robot is modeled as a hyper-redundant, floating-base manipulator. The constrained kinematics arise from the presence of obstacles. Three different kinds of frames are distinguished:

1. Inertial frame: $\mathcal{F}^{\text{inertial}}(x, y)$,
2. Contact frames: $\mathcal{F}_i^{\text{c}},\ \forall i \in \{1, \dots, n_{\text{c}}\}$,
3. Task frames: $\mathcal{F}_i^{\text{t}}$, rotated by $\phi_{\text{c},n_{\text{c}}}$,

where n_{c} is the number of hyper surfaces that constrain the snake robot. The origins of $\mathcal{F}_i^{\text{c}}$ and $\mathcal{F}_i^{\text{t}}$ coincide and the vector $\boldsymbol{r}_{\text{t},i}$ is parallel to the robot link that is in contact with the i-th obstacle. This defines the position vectors with respect to $\theta_{\text{t},i} = \phi_{\text{c},i}$:

$$\boldsymbol{r}_{\text{c},i} = \begin{bmatrix} x_{\text{c},i} & y_{\text{c},i} & 0 \end{bmatrix}^\top \in \mathbb{R}^3 \Rightarrow \boldsymbol{r}_{\text{c}} = \begin{bmatrix} r_{\text{c},1} & r_{\text{c},2} & \cdots & r_{\text{c},n_{\text{c}}} \end{bmatrix} \in \mathbb{R}^{3n_{\text{c}}}, \tag{3.15}$$

$$\boldsymbol{r}_{\text{t},i} = \begin{bmatrix} x_{\text{t},i} & y_{\text{t},i} & \theta_{\text{t},i} \end{bmatrix}^\top \in \mathbb{R}^3 \Rightarrow \boldsymbol{r}_{\text{t}} = \begin{bmatrix} r_{\text{t},1} & r_{\text{t},2} & \cdots & r_{\text{t},n_{\text{c}}} \end{bmatrix} \in \mathbb{R}^{3n_{\text{c}}}. \tag{3.16}$$

The virtual servo coordinates $\boldsymbol{q}_{\text{s}}$ are extended to include the associated distances k_i and angles $\phi_{\text{c},i}$ at which contacts occur, as seen in Equation (3.6). The coordinates of a contact are given through $(x_{\text{c},i}, y_{\text{c},i})$.

The linear and angular velocities of the snake at the moment of contact can be related to $\boldsymbol{q}_\mathrm{e}$ through the Jacobian matrix $\boldsymbol{J}_\mathrm{c}(\boldsymbol{q}_\mathrm{e})$, such that

$$\boldsymbol{v}_{\mathrm{c},i} = \boldsymbol{J}_{\mathrm{c},i}(\boldsymbol{q}_\mathrm{e})\dot{\boldsymbol{q}}_\mathrm{e} \quad \text{or consequently} \quad \dot{\boldsymbol{q}}_\mathrm{e} = \boldsymbol{J}^\dagger_{\mathrm{c},i}(\boldsymbol{q}_\mathrm{e})\boldsymbol{v}_{\mathrm{c},i} \tag{3.17}$$

holds with $\boldsymbol{J}^\dagger_{\mathrm{c},i} := \left(\boldsymbol{J}^\top_{\mathrm{c},i}\boldsymbol{J}_{\mathrm{c},i}\right)^{-1}\boldsymbol{J}^\top_{\mathrm{c},i}$ being the pseudo inverse of the constraint Jacobian

$$\boldsymbol{J}_{\mathrm{c},i}(\boldsymbol{q}_\mathrm{e}) = \begin{bmatrix} \dfrac{\partial x_{\mathrm{c},i}}{\partial q_{\mathrm{e},1}} \cdots \dfrac{\partial x_{\mathrm{c},i}}{\partial q_{\mathrm{e},N}} \\ \dfrac{\partial y_{\mathrm{c},i}}{\partial q_{\mathrm{e},1}} \cdots \dfrac{\partial y_{\mathrm{c},i}}{\partial q_{\mathrm{e},N}} \\ 0 \quad \cdots \quad 0 \end{bmatrix}. \tag{3.18}$$

With that, $\boldsymbol{J}_\mathrm{c} = \begin{bmatrix} \boldsymbol{J}_{\mathrm{c},1} & \boldsymbol{J}_{\mathrm{c},2} & \cdots & \boldsymbol{J}_{\mathrm{c},n_\mathrm{c}} \end{bmatrix} \in \mathbb{R}^{3n_\mathrm{c}\times N}$ can be formulated. Likewise, the task Jacobian can be defined as $\boldsymbol{J}_\mathrm{t} = \begin{bmatrix} \boldsymbol{J}_{\mathrm{t},1} & \boldsymbol{J}_{\mathrm{t},2} & \cdots & \boldsymbol{J}_{\mathrm{t},n_\mathrm{t}} \end{bmatrix} \in \mathbb{R}^{3n_\mathrm{t}\times N}$ with

$$\boldsymbol{J}_{\mathrm{t},i}(\boldsymbol{q}_\mathrm{e}) = \begin{bmatrix} \dfrac{\partial x_{\mathrm{t},i}}{\partial q_{\mathrm{t},1}} \cdots \dfrac{\partial x_{\mathrm{t},i}}{\partial q_{\mathrm{t},N}} \\ \dfrac{\partial y_{\mathrm{t},i}}{\partial q_{\mathrm{t},1}} \cdots \dfrac{\partial y_{\mathrm{t},i}}{\partial q_{\mathrm{t},N}} \\ \dfrac{\partial \theta_{\mathrm{t},i}}{\partial q_{\mathrm{t},1}} \cdots \dfrac{\partial \theta_{\mathrm{t},i}}{\partial q_{\mathrm{t},N}} \end{bmatrix} \tag{3.19}$$

through the task velocities

$$\boldsymbol{v}_{\mathrm{t},i} = \boldsymbol{J}_{\mathrm{t},i}(\boldsymbol{q}_\mathrm{e})\dot{\boldsymbol{q}}_\mathrm{e} \quad \text{or consequently} \quad \dot{\boldsymbol{q}}_\mathrm{e} = \boldsymbol{J}^\dagger_{\mathrm{t},i}(\boldsymbol{q}_\mathrm{e})\boldsymbol{v}_{\mathrm{t},i}. \tag{3.20}$$

Note that at the positions of contact

$$(x_{\mathrm{c},i}, y_{\mathrm{c},i}) = (x_{\mathrm{t},i}, y_{\mathrm{t},i}) \tag{3.21}$$

and the angles at which contacts occur are equal for the task and constraint frames. As stated before in the assumptions, a known path is available. Additionally, the contact points for propulsion are defined a priori. The constraint vector is then expressed by

$$\boldsymbol{p}_i(\boldsymbol{r}_\mathrm{t}) = \boldsymbol{0}, \quad i \in \{1, \ldots, n_\mathrm{c}\}. \tag{3.22}$$

Due to the nature of the definition of the task frames, no velocities are possible in the y-direction and therefore,

$$\dot{y}_{\mathrm{t},i} = 0. \tag{3.23}$$

For the unit normal vectors of the hypersurfaces

$$\boldsymbol{E}_{\mathrm{F}} = \frac{\dfrac{\partial \boldsymbol{p}_i(\boldsymbol{r}_{\mathrm{t}})}{\partial \boldsymbol{r}_{\mathrm{t}}}}{\left\| \dfrac{\partial \boldsymbol{p}_i(\boldsymbol{r}_{\mathrm{t}})}{\partial \boldsymbol{r}_{\mathrm{t}}} \right\|}, \tag{3.24}$$

the following implication holds:

$$\frac{\partial \boldsymbol{p}_i(\boldsymbol{r}_{\mathrm{t}})}{\partial \boldsymbol{r}_{\mathrm{t}}} \frac{\partial \boldsymbol{r}_{\mathrm{t}}}{\partial t} = \boldsymbol{0} \Rightarrow \boldsymbol{E}_{\mathrm{F}} \dot{\boldsymbol{r}}_{\mathrm{t}} = \boldsymbol{0}. \tag{3.25}$$

This is also true for every i-th component of $\boldsymbol{E}_{\mathrm{F}}$ and therefore,

$$\boldsymbol{E}_{\mathrm{F},i} \dot{\boldsymbol{r}}_{\mathrm{t},i} = \begin{bmatrix} \boldsymbol{E}_{\mathrm{F},i,1} & \boldsymbol{E}_{\mathrm{F},i,2} & \boldsymbol{E}_{\mathrm{F},i,3} \end{bmatrix} \begin{bmatrix} \dot{x}_{\mathrm{t},i} \\ \dot{y}_{\mathrm{t},i} \\ \dot{\theta}_{\mathrm{t},i} \end{bmatrix} = 0. \tag{3.26}$$

With Equation (3.23) and Equation (3.26), a contact point velocity constraint is given through

$$\boldsymbol{E}_{\mathrm{F},i} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}. \tag{3.27}$$

This means that $\exists!\ \boldsymbol{E}_{\mathrm{F},i} \wedge r_{\mathrm{t},i},\ \forall i \in \{1, \ldots, n_{\mathrm{c}}\}$, such that

$$\boldsymbol{E}_{\mathrm{F}} = \begin{bmatrix} \boldsymbol{E}_{\mathrm{F},1} & 0_{1\times 3} & \cdots & 0_{1\times 3} \\ 0_{1\times 3} & \boldsymbol{E}_{\mathrm{F},2} & \cdots & 0_{1\times 3} \\ \vdots & & \ddots & \vdots \\ 0_{1\times 3} & \cdots & \cdots & \boldsymbol{E}_{\mathrm{F},n_{\mathrm{c}}} \end{bmatrix} \in \mathbb{R}^{n_{\mathrm{c}} \times 3 n_{\mathrm{c}}}. \tag{3.28}$$

The complement to $\boldsymbol{E}_{\mathrm{F},i}$ is

$$\boldsymbol{E}_{\mathrm{P},i} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.29}$$

and as 'P' indicates, it belongs to the defining directions for the motion control. Analogously,

$$\boldsymbol{E}_{\mathrm{P}} = \begin{bmatrix} \boldsymbol{E}_{\mathrm{P},1} & 0_{2\times 3} & \cdots & 0_{2\times 3} \\ 0_{2\times 3} & \boldsymbol{E}_{\mathrm{P},2} & \cdots & 0_{2\times 3} \\ \vdots & & \ddots & \vdots \\ 0_{2\times 3} & \cdots & \cdots & \boldsymbol{E}_{\mathrm{P},n_{\mathrm{c}}} \end{bmatrix} \in \mathbb{R}^{2 n_{\mathrm{c}} \times 3 n_{\mathrm{c}}}. \tag{3.30}$$

Thus, the construction for $\boldsymbol{E} = \begin{bmatrix} \boldsymbol{E}_\mathrm{P} & \boldsymbol{E}_\mathrm{F} \end{bmatrix}^\top \in \mathbb{R}^{3n_\mathrm{c} \times 3n_\mathrm{c}}$ is completed. The matrix $\boldsymbol{E}$ represents the transformation from the inertial frame to the task frame. Taking a look at the second time derivative

$$\frac{d}{dt}(\boldsymbol{E}_\mathrm{F}\dot{\boldsymbol{r}}_\mathrm{t}) = \boldsymbol{0} \Rightarrow \underbrace{\dot{\boldsymbol{E}}_\mathrm{F}\dot{\boldsymbol{r}}_\mathrm{t}}_{=:\boldsymbol{a}_{\mathrm{r,F}}} + \boldsymbol{E}_\mathrm{F}\ddot{\boldsymbol{r}}_\mathrm{t} = \boldsymbol{0} \tag{3.31}$$

reveals a generalization for $\boldsymbol{E}$, where

$$\boldsymbol{E}\dot{\boldsymbol{r}}_\mathrm{t} = \begin{bmatrix} \boldsymbol{E}_\mathrm{P} \\ \boldsymbol{E}_\mathrm{F} \end{bmatrix} \dot{\boldsymbol{r}}_\mathrm{t} = \begin{bmatrix} \boldsymbol{E}_\mathrm{P}\dot{\boldsymbol{r}}_\mathrm{t} \\ \boldsymbol{0} \end{bmatrix}. \tag{3.32}$$

This holds since the normal component of the velocity is zero, as shown in Equation (3.25). Applying the results of Equation (3.32) to the acceleration in Equation (3.31) yields

$$\boldsymbol{E}\ddot{\boldsymbol{r}}_\mathrm{t} = \begin{bmatrix} \boldsymbol{E}_\mathrm{P} \\ \boldsymbol{E}_\mathrm{F} \end{bmatrix} \ddot{\boldsymbol{r}}_\mathrm{t} = \begin{bmatrix} \boldsymbol{E}_\mathrm{P}\ddot{\boldsymbol{r}}_\mathrm{t} \\ -\boldsymbol{a}_{\mathrm{r,F}} \end{bmatrix}. \tag{3.33}$$

It is found that the acceleration for each contact only dependent of the velocity.

When partitioning the configuration space $\mathbb{C}$, constraints embedded in $\boldsymbol{E}$ are used. Coming back to the initial robotic Equation (3.9), it is found that

$$\boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\ddot{\boldsymbol{q}}_\mathrm{e} + \boldsymbol{h}(\boldsymbol{q}_\mathrm{e}, \dot{\boldsymbol{q}}_\mathrm{e}) = \underbrace{\boldsymbol{\tau}_\mathrm{c}}_{=\boldsymbol{\tau}_\mathrm{P}+\boldsymbol{\tau}_\mathrm{F}} - \underbrace{\boldsymbol{\tau}_\mathrm{F}^*}_{=\boldsymbol{J}_\mathrm{t}^\top \boldsymbol{f}} \tag{3.34}$$

is an extension, where $\boldsymbol{\tau}_\mathrm{P}$ represents the motion control input and $\boldsymbol{\tau}_\mathrm{F}$ the force control input. The constraint force from a contact is represented by $\boldsymbol{\tau}_\mathrm{F}^* = \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{f}$, where $\boldsymbol{f} = \boldsymbol{E}_\mathrm{F}^\top \boldsymbol{f}_\mathrm{F}$ with the force vector at contact $\boldsymbol{f}_\mathrm{F}$. Note that the first three components of $\boldsymbol{\tau}_\mathrm{s}$ are virtual coordinates and therefore, the corresponding torques are zero. If the inverse dynamics controller

$$\boldsymbol{\tau}_\mathrm{P} = \boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\boldsymbol{a}_\alpha + \boldsymbol{h}(\boldsymbol{q}_\mathrm{e}, \dot{\boldsymbol{q}}_\mathrm{e}) \tag{3.35}$$

and the feed-forward term

$$\boldsymbol{\tau}_\mathrm{F} = \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{E}_\mathrm{F}^\top \boldsymbol{a}_\phi \tag{3.36}$$

are inserted into Equation (3.34), the following requirements for a possible controller arise:

$$\boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\ddot{\boldsymbol{q}}_\mathrm{e} + \boldsymbol{h}(\boldsymbol{q}_\mathrm{e}, \dot{\boldsymbol{q}}_\mathrm{e}) = \boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\boldsymbol{a}_\alpha + \boldsymbol{h}(\boldsymbol{q}_\mathrm{e}, \dot{\boldsymbol{q}}_\mathrm{e}) + \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{E}_\mathrm{F}^\top \boldsymbol{a}_\phi - \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{E}_\mathrm{f}^\top \boldsymbol{f}_\mathrm{F}, \tag{3.37}$$

$$\boldsymbol{0} = \boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\boldsymbol{a}_\alpha - \boldsymbol{M}(\boldsymbol{q}_\mathrm{e})\ddot{\boldsymbol{q}}_\mathrm{e} - \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{E}_\mathrm{F}^\top \boldsymbol{a}_\phi + \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{E}_\mathrm{f}^\top \boldsymbol{f}_\mathrm{F}, \tag{3.38}$$

$$\boldsymbol{0} = \boldsymbol{M}(\boldsymbol{q}_\mathrm{e})(\boldsymbol{a}_\alpha + \ddot{\boldsymbol{q}}_\mathrm{e}) - \boldsymbol{J}_\mathrm{t}^\top \boldsymbol{E}_\mathrm{F}^\top (\boldsymbol{a}_\phi + \boldsymbol{f}_\mathrm{F}). \tag{3.39}$$

The parameters $\boldsymbol{a}_\alpha$ and $\boldsymbol{a}_\phi$ are the configuration parameters for this controller. Yoshikawa [Yos87] and Spong et al. [SHV06] chose inverse dynamics to handle the task space controller

$$\boldsymbol{a}_\alpha = \boldsymbol{J}_\mathrm{t}^\dagger (\boldsymbol{a}_\chi - \dot{J}_\mathrm{t}\boldsymbol{q}_\mathrm{e}), \tag{3.40}$$

where

$$\boldsymbol{a}_\chi = \ddot{\boldsymbol{r}}_{\mathrm{t,P}} = \ddot{\boldsymbol{r}}_{\mathrm{t,P}}^d + \boldsymbol{K}_{\mathrm{P}_1}(\boldsymbol{r}_{\mathrm{t,P}}^d - \boldsymbol{r}_{\mathrm{t,P}}) + \boldsymbol{K}_\mathrm{D}(\dot{\boldsymbol{r}}_{\mathrm{t,P}}^d - \dot{\boldsymbol{r}}_{\mathrm{t,P}}) \tag{3.41}$$

is a PD-controller with a feed-forward term from a reference in task frame coordinates for motion control. The motion control task coordinates are represented by $\ddot{\boldsymbol{r}}_{\mathrm{t,P}}$, while $\ddot{\boldsymbol{r}}_{\mathrm{t,P}}^\mathrm{d}$ is a known reference. The matrices $\boldsymbol{K}_{\mathrm{P}_1}$ and $\boldsymbol{K}_\mathrm{D}$ are tunable to individual preferences. Note that when applying $\boldsymbol{a}_\chi$ to Equation (3.39), the transformation to joint space coordinates, as seen in Equation (3.40), needs to be executed first. Finally, the remaining parameter $\boldsymbol{a}_\phi = \boldsymbol{f}_\mathrm{F}^d + \boldsymbol{K}_{\mathrm{P}_2}(\boldsymbol{f}_\mathrm{F}^\mathrm{d} - \boldsymbol{f}_\mathrm{F})$ also consists of a known reference in $\boldsymbol{f}_\mathrm{F}^\mathrm{d}$. Here, $\boldsymbol{f}_\mathrm{F}$ is the force controlled direction by $\boldsymbol{E}_\mathrm{F}$, which is controlled by another tunable proportional gain $\boldsymbol{K}_{\mathrm{P}_2}$.

Once all of these steps are applied, the closed-control loop becomes linear, representing a decoupled system. It simultaneously controls the forces exerted on the obstacles and the motion along the obstacles in the appropriate direction. This approach, as later referenced in Chapter 4, is relying on the dynamics of the system. In contrast to this, another model can be formulated, which is explained in the following section.

3.3.4 Passivity-Based Kinematic HPFC

In the assumptions in Section 3.3.2, the dynamic effects have been deliberately neglected by modeling the snake robot as a quasi-static, passive system. Under these conditions, a static control approach is desirable. West and Asada [WA85] introduced a method for

the design of HPFC controllers, where manipulators are constrained by contact with the environment. Their approach is particularly promising for OAL in snake robots, which must continuously maintain contact with obstacles to achieve propulsion and ensure stable configurations. Since passivity is a fundamental assumption in this work, it is properly defined below.

Definition of Passivity and Dissipative Systems

The following definitions are adopted from KHALIL's book *Nonlinear Systems* [Kha01]. A nonlinear system of the form

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{g}(\boldsymbol{x})\boldsymbol{u}, \quad \boldsymbol{y} = \boldsymbol{h}(\boldsymbol{x}), \tag{3.42}$$

with state $\boldsymbol{x} \in \mathbb{R}^n$, input $\boldsymbol{u} \in \mathbb{R}^m$, and output $\boldsymbol{y} \in \mathbb{R}^m$ is said to be *dissipative* with respect to a supply rate $\boldsymbol{w}(\boldsymbol{u}, \boldsymbol{y})$, if there exists a continuous, non-negative storage function $V(\boldsymbol{x})$ (with $V(\boldsymbol{0}) = 0$), such that for all admissible inputs and for all $t \geq 0$:

$$V(\boldsymbol{x}(t)) - V(\boldsymbol{x}(0)) \leq \int_0^t \boldsymbol{w}(\boldsymbol{u}(s), \boldsymbol{y}(s))\, ds. \tag{3.43}$$

A common choice for the supply rate is

$$\boldsymbol{w}(\boldsymbol{u}(t), \boldsymbol{y}(t)) = \boldsymbol{u}^\top(t)\boldsymbol{y}(t). \tag{3.44}$$

In this case, the system is said to be *passive*. Intuitively, passivity characterizes systems that do not generate energy on their own, meaning they can only store or dissipate energy. Two important subclasses of passive systems are:

1. **Lossless Systems:** These satisfy Equation 3.43 as an equality,

$$V(\boldsymbol{x}(t)) - V(\boldsymbol{x}(0)) = \int_0^t \boldsymbol{u}^\top(s)\boldsymbol{y}(s)\, ds,$$

   indicating that all supplied energy is stored without any net dissipation.

2. **Strictly Passive Systems:** There exists a positive definite function $S(\boldsymbol{x})$, such that
$$V(\boldsymbol{x}(t)) - V(\boldsymbol{x}(0)) < \int_0^t \left[\boldsymbol{u}^\top(s)\boldsymbol{y}(s) - S(\boldsymbol{x}(s))\right] ds.$$
This additional dissipation term $S(\boldsymbol{x})$ guarantees further energy loss, which is crucial for stability.

Relevance in this Work: In the context of HOAL, passivity supports the kinematic control strategy by ensuring that any energy introduced via control inputs is either stored or dissipated rather than amplified. This energy based viewpoint provides a natural framework for simultaneously addressing position and force objectives, a core feature of the HPFC approach utilized in this thesis. However, in this particular study, passivity, and thus stability, is primarily assumed rather than analyzed explicitly. By relying on this assumption, attention can be fully directed toward addressing the main research question stated in Section 1.2, whether HPFC can reliably navigate a simulated snake robot through a predefined obstacle course using OAL.

A major challenge in controlling constrained manipulators arises from two issues:

1. The reduction of available DOF due to contact constraints.
2. The reaction forces that develop at the contacts.

In this context, these reaction forces are not only inevitable but also essential—they are utilized to generate propulsion. Thus, it is critical to partition the joint space into the already introduced complementary subspaces, as motivated in Figure 3.5. The following section is inspired by the work of West and Asada [WA85].

Kinematics and Contact Constraints: Consider a small motion of the end-effector given by
$$\delta\boldsymbol{x} = \boldsymbol{J}\,\delta\boldsymbol{\theta}, \tag{3.45}$$
where $\delta\boldsymbol{x} \in \mathbb{R}^6$ represents the end-effector displacement, $\delta\boldsymbol{\theta} \in \mathbb{R}^m$ the joint displacement, and $\boldsymbol{J} \in \mathbb{R}^{6\times m}$ is the manipulator Jacobian. In the assumed frictionless, quasi-static snake robot model, the work experienced by the end-effector is the sum of the work per-

formed at each joint. Thus, if a force $\boldsymbol{q} \in \mathbb{R}^6$ acts on the end-effector, the corresponding joint torques are given by

$$\boldsymbol{\tau} = \boldsymbol{J}^\top \boldsymbol{q}. \tag{3.46}$$

When the robot is in contact with obstacles, the motion of the joints is further constrained by contact conditions. These constraints are expressed as

$$\boldsymbol{J}_\mathrm{c}\, \delta\boldsymbol{\theta} = \boldsymbol{0}, \tag{3.47}$$

where $\boldsymbol{J}_\mathrm{c}$ is the contact Jacobian mapping joint velocities to the velocity at the contact point. Consequently, allowable joint motions must lie in the null space of $\boldsymbol{J}_\mathrm{c}$.

Projection Operators and Task-Space Decomposition: Following West and Asada [WA85], two projectors in the joint space are defined:

$$j\boldsymbol{P}_\theta = \boldsymbol{I} - \boldsymbol{J}_\mathrm{c}^\dagger \boldsymbol{J}_\mathrm{c}, \tag{3.48}$$

$$j\boldsymbol{P}_\mathrm{F} = \boldsymbol{J}_\mathrm{c}^\dagger \boldsymbol{J}_\mathrm{c}, \tag{3.49}$$

where $\boldsymbol{J}_\mathrm{c}^\dagger$ is the generalized pseudo-inverse of $\boldsymbol{J}_\mathrm{c}$. The projector $j\boldsymbol{P}_\theta$ maps any joint motion onto the allowable motion subspace (which satisfies the contact constraint), while $j\boldsymbol{P}_\mathrm{F}$ projects to the complementary force space. To further refine the control objectives, two sets of essential task variables are defined:

1. *Essential position variables*, which capture the desired motion required for the task, e.g., tangential displacements along obstacles.
2. *Essential force variables*, which represent the required contact forces needed for consistent propulsion, e.g., normal reaction forces.

Let $\boldsymbol{E}_\mathrm{P}$ and $\boldsymbol{E}_\mathrm{F}$ be full-column rank matrices selecting the corresponding components from the task-space. Their orthogonal projectors are given by:

$$\boldsymbol{P}_\mathrm{ep} = \boldsymbol{E}_\mathrm{P} \left(\boldsymbol{E}_\mathrm{P}^\top \boldsymbol{E}_\mathrm{P} \right)^{-1} \boldsymbol{E}_\mathrm{P}^\top, \tag{3.50}$$

$$\boldsymbol{P}_\mathrm{ef} = \boldsymbol{E}_\mathrm{F} \left(\boldsymbol{E}_\mathrm{F}^\top \boldsymbol{E}_\mathrm{F} \right)^{-1} \boldsymbol{E}_\mathrm{F}^\top. \tag{3.51}$$

HPFC Controller Design

The objective of the HPFC controller is to compute a joint motion command $\Delta\boldsymbol{\theta}_{\mathrm{d}}$ that achieves the desired task while respecting contact constraints. A desired joint motion $\Delta\boldsymbol{\theta}_{\mathrm{r}}$ is generated, for example, by a high-level planner. It is filtered by a so-called hybrid projection operator $\boldsymbol{F}$, such that

$$\Delta\boldsymbol{\theta}_{\mathrm{d}} = \boldsymbol{F}\,\Delta\boldsymbol{\theta}_{\mathrm{r}}. \tag{3.52}$$

The filter $\boldsymbol{F}$ must satisfy two key conditions:

1. *Constraint Preservation:* The filtered command must lie in the allowable motion subspace

$$\boldsymbol{F} = j\boldsymbol{P}_{\theta}\,\boldsymbol{F}, \tag{3.53}$$

ensuring that the contact constraint (3.47) is never violated.

2. *Task Preservation:* The essential motion required for the task must be preserved:

$$\boldsymbol{P}_{\mathrm{ep}}\,\boldsymbol{F} = \boldsymbol{P}_{\mathrm{ep}}. \tag{3.54}$$

A general solution that meets these requirements is

$$\boldsymbol{F} = j\boldsymbol{P}_{\theta} + j\boldsymbol{P}_{\mathrm{F}}\,\boldsymbol{X}, \tag{3.55}$$

with $\boldsymbol{X}$ determined by imposing the condition in (3.54):

$$\boldsymbol{P}_{\mathrm{ep}}\,j\boldsymbol{P}_{\mathrm{F}}\,\boldsymbol{X} = \boldsymbol{P}_{\mathrm{ep}} - \boldsymbol{P}_{\mathrm{ep}}\,j\boldsymbol{P}_{\theta}. \tag{3.56}$$

A minimum-norm solution for $\boldsymbol{X}$ is obtained by

$$\boldsymbol{X} = (\boldsymbol{P}_{\mathrm{ep}}\,j\boldsymbol{P}_{\mathrm{F}})^{\dagger}\,(\boldsymbol{P}_{\mathrm{ep}} - \boldsymbol{P}_{\mathrm{ep}}\,j\boldsymbol{P}_{\theta})\,, \tag{3.57}$$

so that the final projection filter becomes

$$\boldsymbol{F} = j\boldsymbol{P}_{\theta} + j\boldsymbol{P}_{\mathrm{F}}\,(\boldsymbol{P}_{\mathrm{ep}}\,j\boldsymbol{P}_{\mathrm{F}})^{\dagger}\,(\boldsymbol{P}_{\mathrm{ep}} - \boldsymbol{P}_{\mathrm{ep}}\,j\boldsymbol{P}_{\theta})\,. \tag{3.58}$$

Implementation Requirements for the Snake Robot: For the simulated snake robot, which operates under quasi-static conditions, the kinematic HPFC is implemented as follows:

1. A high-level path planner produces a desired joint motion $\Delta\boldsymbol{\theta}_{\mathrm{r}}$.
2. The projection filter $\boldsymbol{F}$ (Equation (3.58)) is applied to obtain $\Delta\boldsymbol{\theta}_{\mathrm{d}}$, ensuring that only those joint motions permissible by the contact constraints are executed.
3. Due to the nature of the contact, the controller inherently decouples the motion (position) and reaction (force) components, thereby fulfilling the task objectives.

Summary: This static (passivity-based) controller offers several advantages:

1. *Simplicity:* By neglecting dynamic effects, the controller design is simplified, which is appropriate for the slow, quasi-static movements of the snake robot.
2. *Constraint Satisfaction:* The use of projection operators guarantees that joint motions do not violate contact conditions, preserving both task performance and structural integrity.
3. *Task Decoupling:* The separation into essential position and force variables allows for independent tuning of motion and force objectives, which is critical for OAL.

The HPFC explained in this section provides a solid framework for controlling snake robots in environments where contact constraints are essential for propulsion. This approach leverages the passive nature of the system and employs kinematic projections to maintain task performance while strictly enforcing contact constraints. Additionally, it has been discussed to derive a controller that guarantees task execution under contact constraints. By filtering the desired joint motions using projection operators that partition the joint space into motion and force subspaces, the controller ensures that the snake robot moves in accordance with both its task requirements and the imposed environmental constraints.

The following Chapter 4 features the steps taken for the implementation of the proposed HPFC into the simulation framework of *SimSerpent*.

4 Integration of Hybrid Position and Force Control (HPFC) in *SimSerpent*

In this chapter, the first steps for deriving the Dynamic Hybrid Position and Force Controller, or DHPFC for short, are given [Yos87]. The realizations are then used to motivate the use of the kinematic HPFC approach introduced in Section 3.3.4. The original *SimSerpent* framework contains redundant features that distract from the actual objective. The adjustments are explained in the following section.

4.1 Simulation Setup

For effective testing, the original *SimSerpent* simulator is adapted to isolate and evaluate only the aspects relevant to the functionality of the controller. This involves reconfiguring the simulation environment to minimize complexity while preserving key features required for Hybrid Obstacle-Aided Locomotion (HOAL) .

Figure 4.1 presents a concise flowchart of the resulting simulation framework. Beginning with a single configuration file, whose parameters propagate throughout the system and ensure centralized robot, environment, and control settings. A dedicated model builder then converts the body descriptions produced by the body generation modules into a MJCF `xml` file. MoJoCo loads the file, executes the physics simulation, and sets up appropriate callbacks. Sub-modules communicate constantly with the main simulation and can provide control commands, initialization settings, and log data for subsequent analysis.

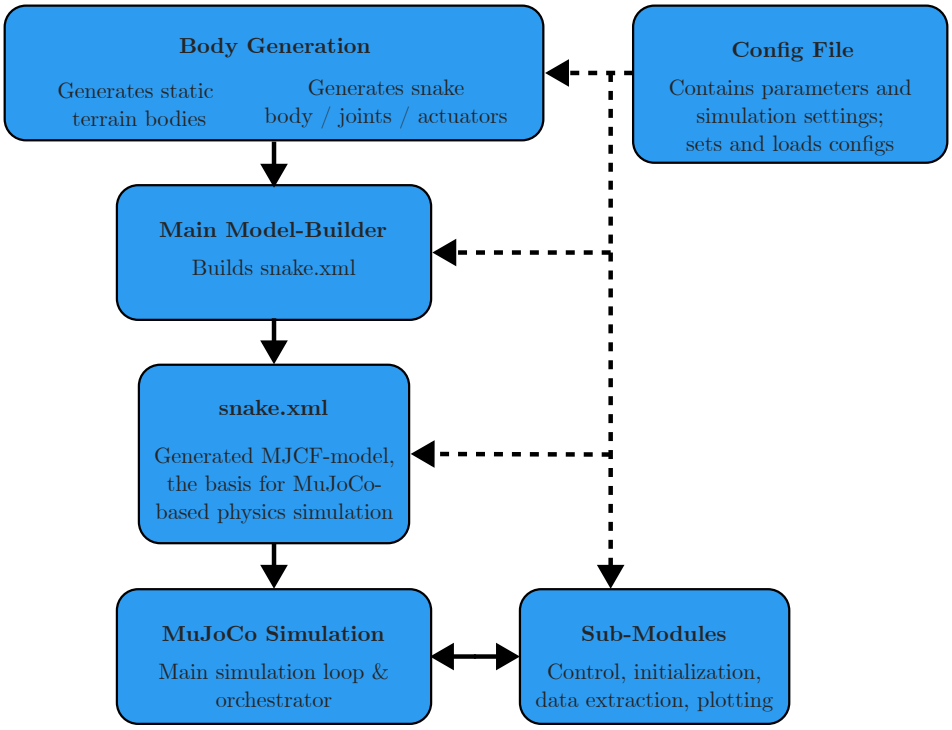


Figure 4.1: Flowchart of *SimSerpent* modules.

The test setup chosen for this work consists of a simple 3-link configuration, as illustrated in Figure 4.2. Joint relations, masses, positions, and dimensions of the bodies are specified and can be adjusted in the configuration file. The robot aims to perform a rocking motion, alternating between forward and backward movement within a confined arrangement of four obstacles.

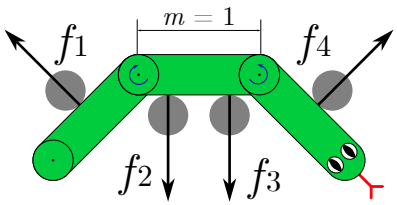


Figure 4.2: Minimal 3-link snake configuration used for controller validation.

Figure 4.3 shows the corresponding setup within the simulation environment. All assumptions described in Section 3.3.2 are taken into account in the simulation, with the exception of small deviations in friction and non-penetration of the bodies. These values are small by default and can be manually disabled. However, they ensure that MuJoCo's physics solver produces realistic results. A dedicated sub-module also provides

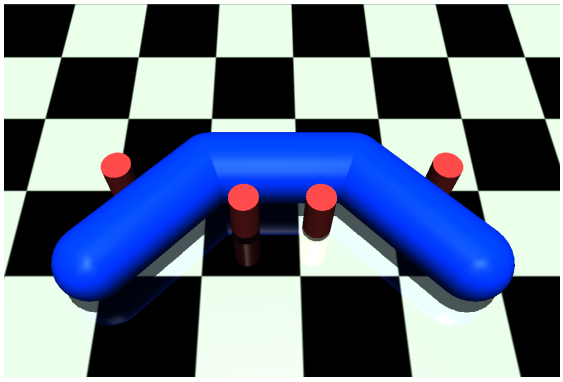


Figure 4.3: Preview of the simulated environment for a 3-link snake robot.

an idealized joint motion reference trajectory $\boldsymbol{\theta}_{\mathrm{r}}(t)$, which is assumed to be free of error. In addition to basic contact detection between the snake body and obstacles, the controller has access to exact contact point information for performance monitoring. This requires continuous access to the spatial relationships between obstacles and the robot body frames. To ensure this information is handled correctly, an error free initialization is guaranteed. This initialization process is described in the following section.

4.1.1 Initialization of the Snake Configuration and Obstacle Placement

As stated in Section 3.3.2, the snake robot is initialized without error at $t = 0$. In the chosen scenario, the simulated snake robot consists of three links. Each link is defined by two joint positions, specifying its start and end position. The links are connected by aligning the end joint position of one link with the start joint position of the next link. Following this logic, snakes of arbitrary length can be constructed. Exceptions are the head and tail links, whose respective start and end joints are not connected to any other link. These are referred to as *virtual joint positions*.

The snake is constructed starting from the tail. The virtual tail joint is placed at $[x_{\mathrm{vj}}, y_{\mathrm{vj}}] = [0, 0]$. From this point, the next joint is positioned at an angle of $\theta_{0,1} = \frac{\pi}{4}\,\mathrm{rad} = 45°$ above the tail joint, at a distance of one link length m. The second joint is placed by shifting the previous joint position along the positive x-axis by m, therefore $\theta_{0,2} = 0°$. Lastly, the third joint is positioned at $\theta_{0,3} = -\frac{\pi}{4}\,\mathrm{rad} = -45°$, bringing the y-

coordinate of the virtual head joint back to zero. This sequence results in a symmetrical C-shape for the three-link snake, as it can be seen in Figures 4.2 and 4.3.

Simultaneously to the snake configuration being is initialized, the obstacles are spawned. They are placed orthogonally with respect to the width of the snake, ensuring immediate contact upon initialization. The obstacle positions are chosen symmetrically for simplicity and to guarantee form closure once a constant torque is applied to the two internal joints of the snake.

Two-Phase Initialization: There are standardized settings within the simulation environment designed to prevent vibrations and make contacts more realistic. A certain penetration depth between objects can be set when creating the snake body and obstacles. In order not to violate the assumptions, this is only done minimally, resulting in no immediate contact upon snake robot spawning. However, in order to provide the controller with the contact points, a two-phase plan is designed, which is tested in the following Chapter 5.

In phase A, a target movement is commended. No direct force command is given, since this requires contacts to begin with. This leads to immediate contact and if applied symmetrically, a balance of forces is achieved. In theory this establishes the contacts and thereby increases the stability of the simulation.

This is followed by phase B, in which other tests can be carried out. It must be shown that the transition between the two phases can take place without a violation of the assumptions. With the snake robot initialized correctly a trajectory can be set for phase B. The following Section 4.1.2 explains how a simple rocking motion trajectory can be derived.

4.1.2 Trajectory Construction

The desired joint angle trajectory and the corresponding position trajectory used in the simulation are illustrated in Figure 4.4.

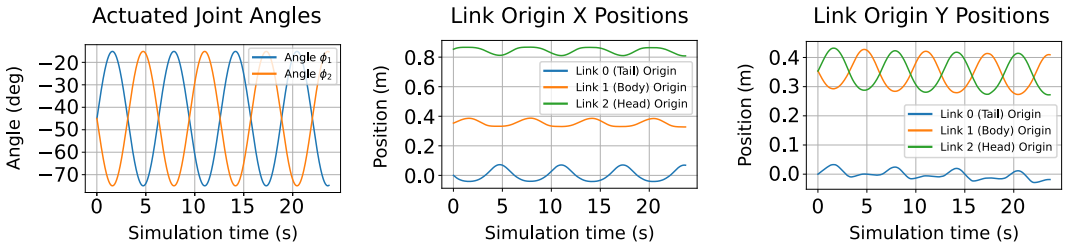


Figure 4.4: Desired joint angle and position trajectories used for a rocking motion.

While both trajectories are visualized for illustrative purposes, only the joint angle reference $\boldsymbol{\theta}_{\mathrm{r}}(t)$ is provided as an input to the controller. The equation for this oscillation pattern, defining the target angle $\theta_{\mathrm{r},i}(t)$ for each actuated joint $i \in \{1, 2\}$ at time $t \in [0, T]$, is

$$\theta_{\mathrm{r},i}(t) = \theta_{0,i} + A_\theta \sin(\omega_{\mathrm{t}} t + (i-1)\Delta\phi_{\mathrm{s}}), \tag{4.1}$$

where, the center angle $\theta_{0,i} = -\frac{\pi}{4}\,\mathrm{rad} = -45°$ is applied to each joint for the specific motion presented in Figure 4.4. These parameters are chosen to align with the initialization angle of the snake robot in this configuration. The amplitude $A_\theta = \frac{\pi}{6}\,\mathrm{rad} = 30°$, the temporal frequency $\omega_{\mathrm{t}} = 1.0\,\mathrm{rad/s}$, and the spatial phase difference between adjacent actuated joints $\Delta\phi_{\mathrm{s}} = \pi\,\mathrm{rad} = 180°$ can be chosen to individual preference. The spatial phase difference causes the two actuated joints to move out of phase, effectively flipping the sign of $\theta_{\mathrm{r},i}(t)$, which creates a rocking motion. The index i refers to the number of actuated joints e.g., $i = 1$ for the first actuated joint angle ϕ_1 and $i = 2$ for the second actuated joint angle ϕ_2.

This presents a universal setup that can be be used for testing several control approaches. The following Section 4.2 derives the dynamic equations for a snake robot in this configuration.

4.2 Dynamic Equations of a Snake Robot

Consider the planar 3-link snake robot composed of rigid links of equal length m and mass M, connected by two revolute joints, as shown in Figure 4.2. The system evolves in an unconstrained, frictionless 2D-plane, for now in the absence of obstacles. Because the snake is located in the (x, y)-plane , no potential energy is present in this model ($P(\boldsymbol{q}) = 0 \Rightarrow \boldsymbol{G}(\boldsymbol{q}) = \mathbf{0}$). The configuration of the robot is described by the generalized coordinates

$$\boldsymbol{q}_\mathrm{s} = \begin{bmatrix} x & y & \phi_0 & \phi_1 & \phi_2 \end{bmatrix}^\top,$$

where (x, y) is the global position of the tail (start of link 1), $\phi_0 = \theta_1$ is the global orientation of the first link, and (ϕ_1, ϕ_2) are the relative angles at the joints between links, as depicted in Figure 3.4. The absolute orientation angles of the three links are defined as:

$$\begin{aligned} \theta_1 &= \phi_0, \\ \theta_2 &= \phi_0 + \phi_1, \\ \theta_3 &= \phi_0 + \phi_1 + \phi_2. \end{aligned}$$

To simplify directional expressions, the unit vector along a link and its orthogonal complement are denoted by:

$$\mathbf{R}(\theta_i) := \begin{bmatrix} \cos\theta_i \\ \sin\theta_i \end{bmatrix}, \quad \mathbf{R}_\perp(\theta_i) := \begin{bmatrix} -\sin\theta_i \\ \cos\theta_i \end{bmatrix}.$$

Each link's center of mass (COM) is located at its midpoint. The COM velocities can then be derived, using the chain rule and the time derivatives of the orientation angles.

Link 1 (Tail link): The CON lies at a distance of $\frac{m}{2}$ in direction $\mathbf{R}(\theta_1)$ from the base position (x, y) and rotates with angular velocity $\dot{\theta}_1 = \dot{\phi}_0$. The translational velocity of the COM is therefore:

$$\dot{\boldsymbol{r}}_1 = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \frac{m}{2}\dot{\phi}_0\, \mathbf{R}_\perp(\theta_1),$$

which yields

$$\boldsymbol{v}_1^2 = \dot{x}^2 + \dot{y}^2 + \frac{m^2}{4}\dot{\phi}_0^2 + m\dot{\phi}_0(-\dot{x}\sin\theta_1 + \dot{y}\cos\theta_1).$$

Link 2 (Middle link): The COM lies at m along $\mathbf{R}(\theta_1)$ and $\frac{m}{2}$ along $\mathbf{R}(\theta_2)$. Its velocity includes both $\dot{\phi}_0$ and $\dot{\phi}_1$:

$$\dot{\boldsymbol{r}}_2 = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + m\dot{\phi}_0 \, \mathbf{R}_\perp(\theta_1) + \frac{m}{2}(\dot{\phi}_0 + \dot{\phi}_1) \, \mathbf{R}_\perp(\theta_2).$$

This leads to:

$$\begin{aligned} \boldsymbol{v}_2^2 = \dot{x}^2 + \dot{y}^2 + m^2\dot{\phi}_0^2 + \frac{m^2}{4}(\dot{\phi}_0 + \dot{\phi}_1)^2 \\ + 2m\dot{\phi}_0(-\dot{x}\sin\theta_1 + \dot{y}\cos\theta_1) \\ + m(\dot{\phi}_0 + \dot{\phi}_1)(-\dot{x}\sin\theta_2 + \dot{y}\cos\theta_2) \\ + m^2\dot{\phi}_0(\dot{\phi}_0 + \dot{\phi}_1)\cos(\theta_1 - \theta_2). \end{aligned}$$

Link 3 (Head link): The COM lies at m along $\mathbf{R}(\theta_1)$, m along $\mathbf{R}(\theta_2)$, and $\frac{m}{2}$ along $\mathbf{R}(\theta_3)$. The velocity requires all angular velocities:

$$\dot{\boldsymbol{r}}_3 = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + m\dot{\phi}_0 \, \mathbf{R}_\perp(\theta_1) + m(\dot{\phi}_0 + \dot{\phi}_1) \, \mathbf{R}_\perp(\theta_2) + \frac{m}{2}(\dot{\phi}_0 + \dot{\phi}_1 + \dot{\phi}_2) \, \mathbf{R}_\perp(\theta_3).$$

Finally, this results in:

$$\begin{aligned} \boldsymbol{v}_3^2 = \dot{x}^2 + \dot{y}^2 + m^2\dot{\phi}_0^2 + m^2(\dot{\phi}_0 + \dot{\phi}_1)^2 + \frac{m^2}{4}(\dot{\phi}_0 + \dot{\phi}_1 + \dot{\phi}_2)^2 \\ + 2m\dot{\phi}_0(-\dot{x}\sin\theta_1 + \dot{y}\cos\theta_1) \\ + 2m(\dot{\phi}_0 + \dot{\phi}_1)(-\dot{x}\sin\theta_2 + \dot{y}\cos\theta_2) \\ + m(\dot{\phi}_0 + \dot{\phi}_1 + \dot{\phi}_2)(-\dot{x}\sin\theta_3 + \dot{y}\cos\theta_3) \\ + 2m^2\dot{\phi}_0(\dot{\phi}_0 + \dot{\phi}_1)\cos(\theta_1 - \theta_2) \\ + m^2(\dot{\phi}_0 + \dot{\phi}_1)(\dot{\phi}_0 + \dot{\phi}_1 + \dot{\phi}_2)\cos(\theta_2 - \theta_3) \\ + 2m^2\dot{\phi}_0(\dot{\phi}_0 + \dot{\phi}_1 + \dot{\phi}_2)\cos(\theta_1 - \theta_3). \end{aligned}$$

In addition to the translational motion, each link undergoes a planar rotation about its center of mass. The angular velocities of the individual links are obtained by differentiating their absolute orientation angles:

$$\begin{aligned}\boldsymbol{\omega}_1 &= \dot{\theta}_1 = \dot{\phi}_0,\\ \boldsymbol{\omega}_2 &= \dot{\theta}_2 = \dot{\phi}_0 + \dot{\phi}_1,\\ \boldsymbol{\omega}_3 &= \dot{\theta}_3 = \dot{\phi}_0 + \dot{\phi}_1 + \dot{\phi}_2.\end{aligned}$$

Each link is modeled as a thin, homogeneous rod of length m and mass M. The moment of inertia about its COM is therefore:

$$I = \frac{1}{12} M m^2.$$

Total Kinetic Energy: The total kinetic energy K consists of the translational and rotational contributions from all three links:

$$K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_{i=1}^{3} \left(\frac{1}{2} M \boldsymbol{v}_i^2 + \frac{1}{2} I \boldsymbol{\omega}_i^2 \right).$$

Substituting the expressions for $\boldsymbol{v}_1^2, \boldsymbol{v}_2^2, \boldsymbol{v}_3^2$, and $\boldsymbol{\omega}_i$ yields

$$\begin{aligned}K(\boldsymbol{q}, \dot{\boldsymbol{q}}) = {} & \frac{1}{2} M (\dot{x}^2 + \dot{y}^2) + \frac{1}{2} I \dot{\phi}_0^2 \\ & + \frac{1}{2} M \left(\dot{x}^2 + \dot{y}^2 + \frac{m^2}{4} \dot{\phi}_1^2 + m \dot{\phi}_1 (-\dot{x} \sin\theta_2 + \dot{y} \cos\theta_2) \right) + \frac{1}{2} I (\dot{\phi}_0 + \dot{\phi}_1)^2 \\ & + \frac{1}{2} M \Bigg(\dot{x}^2 + \dot{y}^2 + m^2 \dot{\phi}_1^2 + \frac{m^2}{4} (\dot{\phi}_1 + \dot{\phi}_2)^2 \\ & \quad - m^2 (\dot{\phi}_1^2 + \dot{\phi}_1 \dot{\phi}_2) \cos(\theta_2 - \theta_3) + 2 m \dot{\phi}_1 (-\dot{x} \sin\theta_2 + \dot{y} \cos\theta_2) \\ & \quad + m (\dot{\phi}_1 + \dot{\phi}_2)(-\dot{x} \sin\theta_3 + \dot{y} \cos\theta_3) \Bigg) + \frac{1}{2} I (\dot{\phi}_0 + \dot{\phi}_1 + \dot{\phi}_2)^2.\end{aligned}$$

This expression defines the complete kinetic energy of the snake robot as a function of the generalized coordinates $\boldsymbol{q}$ and their time derivatives $\dot{\boldsymbol{q}}$. This forms the basis for the dynamic equations derived in the following section, using the Euler-Lagrange formalism.

Euler-Lagrange Formalism and Dynamic Model

To derive the equations of motion, the Euler–Lagrange formalism is applied to the kinetic energy of the system. Since the robot is assumed to move in a horizontal plane without potential energy contributions, the Lagrangian $\mathcal{L}(q, \dot{q})$ reduces to the kinetic energy:

$$\mathcal{L}(q, \dot{q}) = K(q, \dot{q}).$$

The Euler-Lagrange equation for each generalized coordinate q_i is given by

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = Q_i,$$

where Q_i represents the generalized force acting along q_i. In this model, only the internal joint coordinates ϕ_1 and ϕ_2 are actuated through the torques τ_1 and τ_2, while the base coordinates x, y and ϕ_0 remain unactuated:

$$\boldsymbol{Q} = \begin{bmatrix} 0 & 0 & 0 & \tau_1 & \tau_2 \end{bmatrix}^\top = \boldsymbol{U}\boldsymbol{\tau}.$$

The resulting dynamic equations are cast into the standard robotic form:

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

where:

$\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{5\times5}$	is the symmetric, positive-definite mass matrix derived from the second-order partial derivatives of $\mathcal{L}(\boldsymbol{q}, \dot{\boldsymbol{q}})$,
$\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) \in \mathbb{R}^{5\times5}$	represents the Coriolis and centrifugal terms, and
$\boldsymbol{U} \in \mathbb{R}^{5\times2}$	maps joint torques to the generalized force space. It has the form $\boldsymbol{U} = \begin{bmatrix} \mathbf{0}_{3\times2} \\ \mathbf{I}_{2\times2} \end{bmatrix}$.

The elements of the mass matrix are

$$\boldsymbol{M}_{ij}(\boldsymbol{q}) = \frac{\partial^2 K(\boldsymbol{q}, \dot{\boldsymbol{q}})}{\partial \dot{q}_i \partial \dot{q}_j},$$

while the Coriolis matrix is assembled using the Christoffel symbols

$$\boldsymbol{C}_{ij}(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \sum_{k=1}^{5} \frac{1}{2} \left(\frac{\partial \boldsymbol{M}_{ij}(\boldsymbol{q})}{\partial q_k} + \frac{\partial \boldsymbol{M}_{ik}(\boldsymbol{q})}{\partial q_j} - \frac{\partial \boldsymbol{M}_{kj}(\boldsymbol{q})}{\partial q_i} \right) \dot{q}_k.$$

The resulting expressions define the full equations of motion of the floating-base snake robot in terms of its physical parameters and joint configurations.

The detailed results of $\boldsymbol{M}(\boldsymbol{q})$ and $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ were calculated but are too large to prested here. When compared to the 2-DOF planar manipulator showed in Chapter 3, Section 3.1, it is eminent that this procedure is not feasible analytically, since the inverse of $\boldsymbol{M}(\boldsymbol{q})$ would have to be calculated. Furthermore, this 5-DOF system is one of the simplest configurations to present this concept. Longer snake configurations would quickly reach limits of computational power. In this specific example, it may be computable with numerical methods but this approach is definitely not easily scalable. This motivates the use of a physics engine and a model-free and kinematic approach to simulate this system, effectively neglecting the analytically derived dynamics of the snake. In real-world applications, the effectiveness of such an approach depends on the accuracy of the sensors used in the robot. Since the work in this thesis is build upon a simulation, this factor can be neglected. The following section explains how the kinematic HPFC controller is integrated in and adapted to the simulation framework of *SimSerpent*.

4.3 Controller Implementation in *SimSerpent*

This section details the implementation of the HPFC within the adjusted *SimSerpent* simulation environment. While inspired by common theoretical frameworks for HPFC, such as explicit motion filtering, detailed in West and Asada [WA85] or as explained in Section 3.2.3. Here a practical implementation is found, that adopts a direct torque-based approach. This adaptation is tailored to the specifics of the MuJoCo physics engine [TET12] and aims to provide reliable control over both motion and contact forces for the simulated 3-link snake.

The primary goal remains to enable the robot to perform tasks requiring simultaneous control of its motion and the forces exerted on its environment, particularly within form closure scenarios. The controller processes measured joint states and contact information to compute and apply actuator torques.

4.3.1 Control Architecture Overview

The HPFC operates within the discrete-time simulation loop. At each time step, the following logical operations occur:

1. *State Acquisition:* Retrieve current joint angles θ_i, joint velocities $\dot{\theta}_i$, and active contacts from MuJoCo.
2. *Contact Processing:* Determine geometric normal vectors $\mathbf{n}_i$, contact Jacobian columns $\boldsymbol{J}_{\mathrm{c},i}$, and reconstruct measured normal forces F_i^{measured} from MuJoCo's internal constraint data.
3. *Trajectory Generation:* Generate reference joint angles $\theta_{\mathrm{r},i}(t)$ and velocities $\dot{\theta}_{\mathrm{r},i}(t)$ depending on the phase (e.g. static hold, rocking motion).
4. *Torque Computation:* Compute final actuator torques combining motion tracking and force control: $\boldsymbol{\tau}_{\mathrm{final}} = \boldsymbol{\tau}_{\mathrm{motion}} + \boldsymbol{\tau}_{\mathrm{force}}$.
5. *Actuator Command:* Apply computed torques ($\boldsymbol{\tau}_{\mathrm{final}}$) to actuators.
6. *Data Logging:* Log simulation and control variables for analysis.

The proposed two-phase control strategy is employed by an instantaneous switch after a set time step. Phase A typically focuses on positioning the snake using only the motion control, while phase B can utilize the full HPFC logic. These steps are general, and Section 6.3.2 specifically explains the creation process in the simulation environment. Additionally problems and solutions in the implementation are explained.

4.3.2 HPFC Torque Computation Details

The final control torque $\boldsymbol{\tau}_{\mathrm{final}}$ is a superposition of a motion-related torque component $\boldsymbol{\tau}_{\mathrm{motion}}$ and a force-regulation torque component $\boldsymbol{\tau}_{\mathrm{force}}$.

Jacobian Calculation in the Discrete-Time Simulation: In the continuous-time formulation, the manipulator Jacobian $\boldsymbol{J}$ (or contact Jacobian $\boldsymbol{J}_\mathrm{c}$) is defined as a matrix of partial derivatives, mapping joint velocities $\dot{\boldsymbol{\theta}}$ to task-space velocities $\dot{\boldsymbol{x}}$, as seen in Equations (3.18), (3.19), or generally in (3.45):

$$\boldsymbol{J}(\boldsymbol{\theta}) = \frac{\partial \boldsymbol{f}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{4.2}$$

where $\boldsymbol{f}(\boldsymbol{\theta})$ represents the forward kinematics function mapping joint coordinates $\boldsymbol{\theta}$ to task-space coordinates $\boldsymbol{x}$.

In the employed time-discrete simulation environment MuJoCo [TET12], the Jacobian is not derived through finite differentiation across simulation steps. Instead, at each discrete time step t_k, where control computations are performed, the kinematic Jacobian is computed by MuJoCo based on the *current configuration* of the snake robot.

The Python specific controller in this work interacts with MuJoCo and utilizes the built-in function `mujoco.mj_jac`. For a contact point j on a snake body it is called via: `mujoco.mj_jac(model, data, jacp, jacr, contact_pos, snake_body_id)`, where:

- `model` $\mathcal{M}$ contains the full kinematic and dynamic description of the robot (link lengths, joint axes, inertia, etc.) defined a priori in the MJCF (MuJoCo `xml` Format) model file.
- `data` $\mathcal{D}(t_k)$ holds the current state of the simulation at time t_k, including the generalized positions $\boldsymbol{q}(t_k)$, the joint angles $\boldsymbol{\theta}(t_k)$, and velocities $\dot{\boldsymbol{q}}(t_k)$.
- `contact_pos` $p_{\mathrm{c},j}(t_k)$ is the world-frame position of the contact point.
- `snake_body_id` identifies the specific body part of the snake involved in the contact.

The function `mj_jac` then calculates the updated $3\times N_v$ translational Jacobian $\boldsymbol{J}_\mathrm{p}(\boldsymbol{q}(t_{k+1}))$ (`jacp`) and the $3 \times N_v$ rotational Jacobian $\boldsymbol{J}_\mathrm{r}(\boldsymbol{q}(t_{k+1}))$ (`jacr`) for the specified point on the given body with respect to all N_v generalized velocities of the system. This computation leverages the robot's kinematic tree structure defined in `model` and the current joint positions $\boldsymbol{q}(t_k)$ from `data`. MuJoCo internally performs the necessary differentiations of the forward kinematics equations.

The controller then extracts the columns of $\boldsymbol{J}_{\mathrm{p}}(\boldsymbol{q}(t_k))$ corresponding to the n_a actuated joints of the snake robot. They are used to form the $3 \times n_a$ actuated translational Jacobian $\boldsymbol{J}_{\mathrm{p,act},j}(\boldsymbol{q}(t_k))$ in the controller (in the code referred to as `J_ci` for contact j)

$$\boldsymbol{J}_{\mathrm{p,act},j}(\boldsymbol{q}(t_k)) = \left[\frac{\partial p_{\mathrm{c},j}(\boldsymbol{q}(t_k))}{\partial \boldsymbol{\theta}_1(t_k)}, \ldots, \frac{\partial p_{\mathrm{c},j}(\boldsymbol{q}(t_k))}{\partial \boldsymbol{\theta}_{n_a}(t_k)}\right], \tag{4.3}$$

where $p_{\mathrm{c},j}$ is the position of the contact point j and $\boldsymbol{\theta}_j$ are the actuated joint angles. The Jacobian used in the control law at each discrete time step t_k is an *instantaneous kinematic Jacobian*, analytically derived by the physics engine from the robot model and its current configuration $\boldsymbol{q}(t_k)$. It is not an approximation based on numerical differences over time intervals but rather a precise geometric quantity at that specific instant. This Jacobian then forms the basis for computing the constraint normal Jacobian $\boldsymbol{J}_{\mathrm{N}}$, the force feedback torques, and the motion projection matrix $\boldsymbol{\mathcal{P}}_{\mathrm{motion}}$.

1. Motion Control Component ($\boldsymbol{\tau}_{\mathbf{motion}}$): The motion component tracks the desired joint-space trajectory and involves three logical steps:

(a) Desired Motion Torques ($\boldsymbol{\tau}_{\mathbf{des,pd}}$): A proportional-derivative (PD) law computes reference-tracking torques:

$$\boldsymbol{\tau}_{\mathrm{des,pd}} = \boldsymbol{\mathcal{K}}_{\mathrm{P}}^{\mathrm{motion}}(\boldsymbol{\theta}_{\mathrm{r}}(t) - \boldsymbol{\theta}) + \boldsymbol{\mathcal{K}}_{\mathrm{D}}^{\mathrm{motion}}(\dot{\boldsymbol{\theta}}_{\mathrm{r}}(t) - \dot{\boldsymbol{\theta}}),$$

with diagonal gain matrices $\boldsymbol{\mathcal{K}}_{\mathrm{P}}^{\mathrm{motion}}, \boldsymbol{\mathcal{K}}_{\mathrm{D}}^{\mathrm{motion}}$ and reference states $\boldsymbol{\theta}_{\mathrm{r}}(t), \dot{\boldsymbol{\theta}}_{\mathrm{r}}(t)$ from Equation (4.1).

(b) Constraint-Aware Motion Projection: To avoid conflict with force control, motion torques are projected onto the orthogonal subspace compatible with active contact constraints:

$$\boldsymbol{\mathcal{P}}_{\mathrm{motion}} = \mathbf{I} - \boldsymbol{J}_{\mathrm{N}}^{\top}\left(\boldsymbol{J}_{\mathrm{N}}\boldsymbol{J}_{\mathrm{N}}^{\top} + \mathbf{I}\right)^{-1}\boldsymbol{J}_{\mathrm{N}},$$

where the constraint normal Jacobian $\boldsymbol{J}_{\mathrm{N}}$ stacks rows $\boldsymbol{J}_{\mathrm{N},j} = \mathbf{n}_j^{\top}\boldsymbol{J}_{\mathrm{c},j}$ from each active contact j.

(c) Final Motion Torque: The resulting motion torque is

$$\boldsymbol{\tau}_{\text{motion}} = \boldsymbol{\mathcal{P}}_{\text{motion}}\, \boldsymbol{\tau}_{\text{des,pd}}.$$

During pure positioning phases (e.g., phase A), this projection can be bypassed by using $\boldsymbol{\tau}_{\text{motion}} = \boldsymbol{\tau}_{\text{des,pd}}$, to directly enforce the desired pose.

2. Force Control Component ($\boldsymbol{\tau}_{\text{force}}$): The force component regulates the normal forces at active contacts following these steps:

(a) Force Error Calculation: The normal force error for each contact i is

$$e_{\text{F},i} = F_{\text{ref},i} - F_i^{\text{measured}}, \quad \text{with} \quad F_i^{\text{ref}} = k \cdot F_i^{\text{base}},$$

where k is an activation level and F_i^{base} a baseline target force.

(b) Scalar Force Control Law (PI): A proportional-integral (PI) controller computes a scalar control effort

$$u_{F,i} = K_{\text{P},i}^{\text{force}}\, e_{\text{F},i} + K_{\text{I},i}^{\text{force}} \int_{t_0}^{t_1} e_{\text{F},i}\, dt,$$

with PI gains $\boldsymbol{\mathcal{K}}_{\text{P},i}^{\text{force}}$, $\boldsymbol{\mathcal{K}}_{\text{I},i}^{\text{force}}$, including anti-windup limiting.

(c) Mapping to Joint Torques: The scalar effort maps to joint torques via the transpose of the contact's normal Jacobian row:

$$\boldsymbol{\tau}_{\text{force},i} = \boldsymbol{J}_{\text{N},i}^{\top}\, u_{\text{F},i} = (\mathbf{n}_i^{\top} \boldsymbol{J}_{\text{c},i})^{\top}\, u_{\text{F},i}.$$

This ensures that torques effectively influence normal contact forces.

(d) Summation Over Contacts: The total force regulation torque is

$$\boldsymbol{\tau}_{\text{force}} = \sum_i \boldsymbol{\tau}_{\text{force},i}.$$

3. Final Torque Command: The final actuator torques combine motion and force components:

$$\boldsymbol{\tau}_{\text{final}} = \boldsymbol{\tau}_{\text{motion}} + \boldsymbol{\tau}_{\text{force}}.$$

This approach differs from the HPFC method presented in Section 3.2.3, Equation (3.10), which filters desired motions by constraints. Instead, torque contributions from motion and force control are separately computed and combined, using projections to manage interactions. Thus, motion torques remain compatible with required normal forces. PD (motion) and PI (force) controllers form the core of the HPFC implementation with the controlled snake robot in *SimSerpent*. Specific gain settings and gait parameters are depended on experimental objectives as shown in Chapter 5.

4.3.3 Support Modules for Trajectory and Obstacle Generation

To support the controller integration, additional functionality had to be developed within the *SimSerpent* environment. This includes sub-modules for trajectory generation, environment initialization, and visualization. These tools ensure consistency between the physical configuration of the robot, the motion reference, and the obstacle environment.

For the current example, the simple analytically defined trajectory of Equation (4.1) for periodic rocking is motivated. However, the underlying infrastructure supports more complex trajectory generation like propulsion, static holding, and target positions. Other freely assignable settings include placement of obstacle elements, geometric validation of contact conditions, and visual guidance markers. Such tools are essential for transitioning from simplified tests to more general locomotion scenarios. These extensions are not covered in this chapter but are discussed in detail in Chapter 6, including proposed test scenarios and future integration pathways.

To validate the concepts described in this chapter, the subsequent Chapter 5 aims to present the tests conducted in the simulator.

5 Results

This chapter presents the experimental results obtained from the previously explained adjusted simulation environment, in which the presented Hybrid Position and Force Control (HPFC) strategy has been implemented. This includes static force regulation and dynamic motion while maintaining contact, all within a form closure grasp. The general setup of the 3-link snake robot and the obstacle configuration is detailed in Section 4.1 and can be seen in Figure 4.3. All experiments utilize the kinetic HPFC logic with a two-phase control strategy: an initial positioning phase A, followed by a main control phase B to execute the specific experimental objectives. This chapter serves to illustrate the results and to support the content of the resulting figures with brief descriptions.

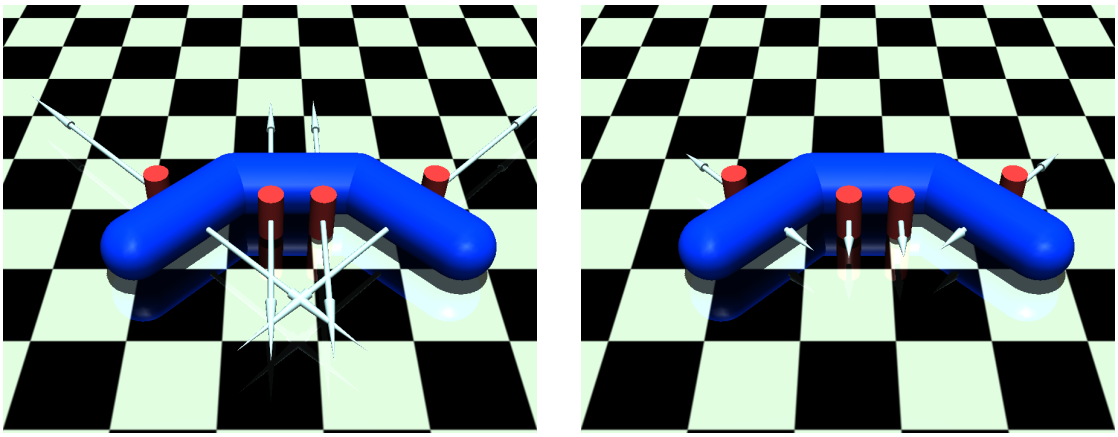
5.1 Static Force Control in Form Closure

The first goal is to evaluate whether the controller maintains a static pose within form closure while tracking different target contact force levels. The experiment involves varying joint torques within the *force subspace* while constraining the motion of the snake. In these tests, the two-phase concept is applied, with the aim of observing whether significant motion or instability occurs. The initial positioning phase A, as described in Section 4.1.1, induces a high contact force amplitude to put the snake robot in position and thereby ensuring form closure (e.g., four contacts). For 2.5 seconds, large contact forces are applied by setting the target joint angles to zero. With this command, the snake undergoes a straightening motion, reaching the predefined torque maximum $\tau_{\text{contact}} = 12$ N. This ensures that, for the second phase B, the decoupled force control has true contacts as a reference. In this phase, the motion control parameters are set to

$\mathbf{K}_\mathrm{p} = \mathbf{0}, \mathbf{K}_\mathrm{d} = \mathbf{0}$, resulting in $\tau_\mathrm{des} = 0$. Only the set torque $\tau_\mathrm{contact} = 4$ N is held as part of the force control command.

5.1.1 Symmetric Static Pose Scenario

Figure 5.1 shows two static configurations of the snake in contact with the environment. Contact vectors are depicted in white, including both action and reaction forces. The left image corresponds to the first 2.5 seconds (phase A) of the pure position control, and the right image shows the configuration for the remainder of the simulation (phase B), which is pure force control.



(a) Phase A: Initialization with high contact forces.

(b) Phase B: Snake after instantaneous reduction of contact force.

Figure 5.1: Visualization of a snake robot in a stable static form closure configuration while undergoing pure force control with switched target force references.

Figure 5.2 presents the quantitative data from the experiment. The plot titled "Sum of Snake-Obstacle Contact Force Magnitudes" in the upper left corner exhibits two plateau regions, each associated with a different commanded target force. An initial transient spike is visible prior to the first plateau. After this spike, the values stabilize around the corresponding commanded force levels. The "Actuated Joint Angles" also exhibit an initial spike, which then settles at $-45°$ throughout the remainder of the simulation. The number of "Snake-Obstacle Contacts" remains constantly at four. Additionally, the "Link Origin X/Y Positions" hold a constant level during the entire simulation time. The "Link Origin Y Positions" only depicts two lines, since the orange "Body Origin" position

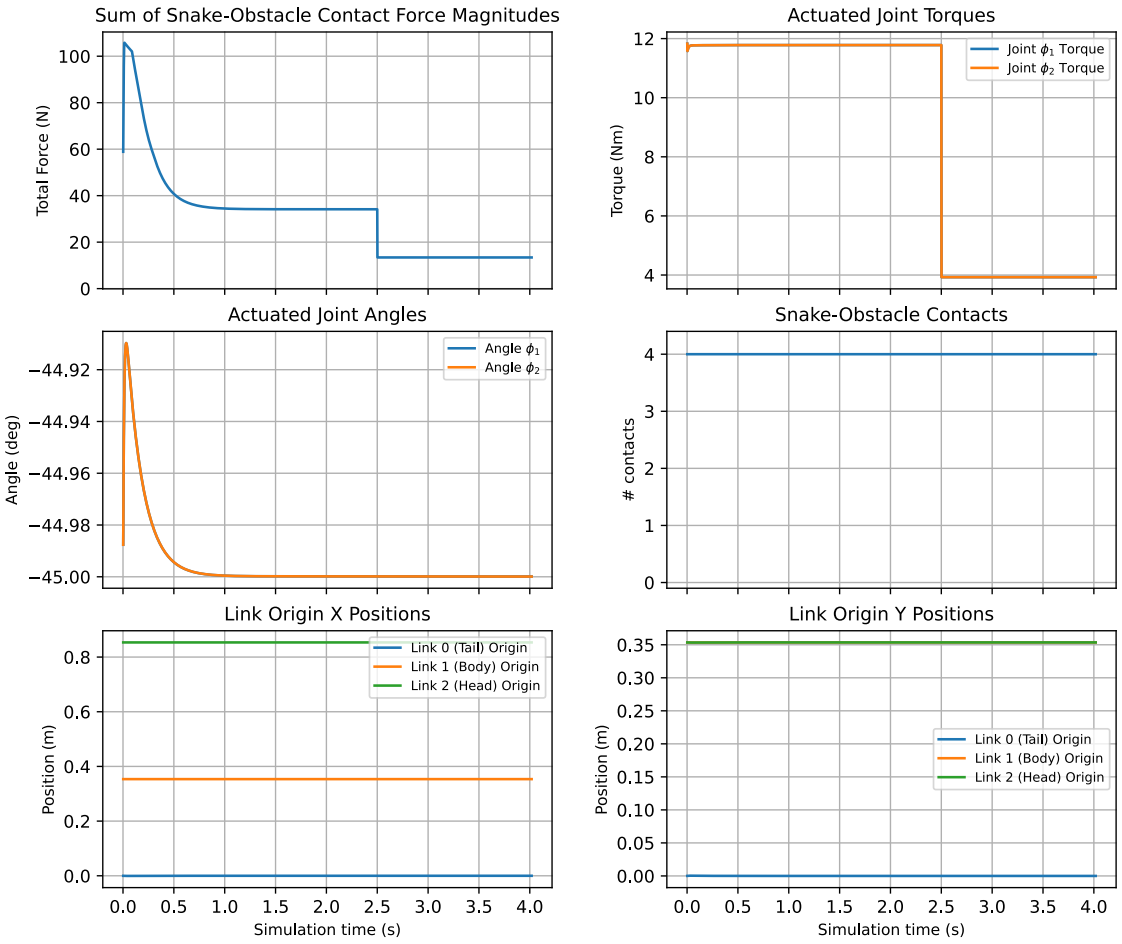


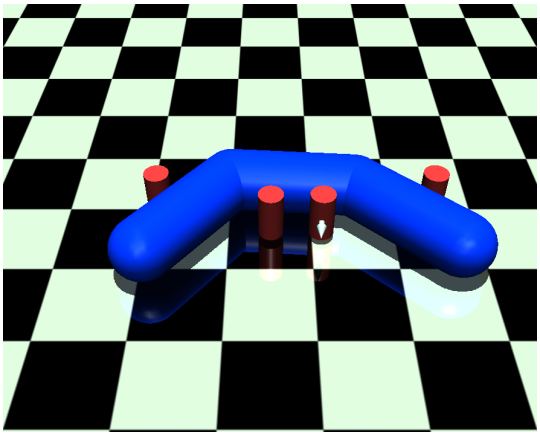
Figure 5.2: Experimental results for static form closure with switched target contact force references.

is overlaid by the green "Head Origin" line. The "Actuated Joint Torques" exhibit two distinct plateaus of 12 Nm and of 4 Nm. The switch occurs at 2.5 seconds.

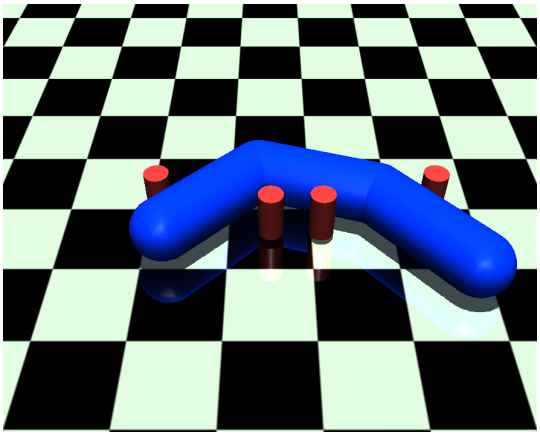
5.1.2 Target Asymmetric Static Pose Scenario

In addition to tests focused on force control within form closure, an experiment is conducted to observe the HPFC motion control behavior. The snake is commanded to move from its initial bent configuration to a different, non-symmetric static pose $\left[\theta_1^{\text{target}} = -1.0,\ \theta_2^{\text{target}} = -0.5\right]$ and maintain it. For this test, phase A lasts 1.5 seconds and targets the new pose. In Section 4.1.1 it is stated that no force command is applied

in phase A, but the opposite is done here to evaluate the control performance. In phase B, the motion gains are set to zero, and a low force command of 0.05 N is applied while the pose is maintained. The setup for this test can be seen in Figure 5.3, where the snake can be seen transitioning from the pose on the left to the target pose on the right.



(a) Snake transitioning to a target non-symmetric pose (view 1).



(b) Snake holding the target non-symmetric pose (view 2).

Figure 5.3: Visualization of a snake robot achieving and holding a commanded non-symmetric static target pose.

Figure 5.4 shows that the "Actuated Joint Angles" change during phase A and remain approximately constant during phase B, at levels consistent with the target values. The same can be observed in "Link Origin X/Y Positions". In phase A, short and high spikes in the "Sum of Snake-Obstacle Contact Force Magnitudes" appear. This can also be seen in "Snake-Obstacle Contacts" and partially in "Actuated Joint Torques". For phase B, the "Snake-Obstacle Contacts" remain constantly at three. The "Actuated Joint Torques" show flat and constant values in this phase as well. The contact force magnitude spikes again after the transition between the phases and finally reaches a constant level at approximately 2.75 seconds.

5.2 Dynamic Motion with Modulated Force Control

The second goal is to demonstrate the robot moving back and forth within a configuration exhibiting form closure, while simultaneously controlling contact forces at different activation levels $k \in \mathbb{R}$. As shown previously, phase A settles initial dynamic effects

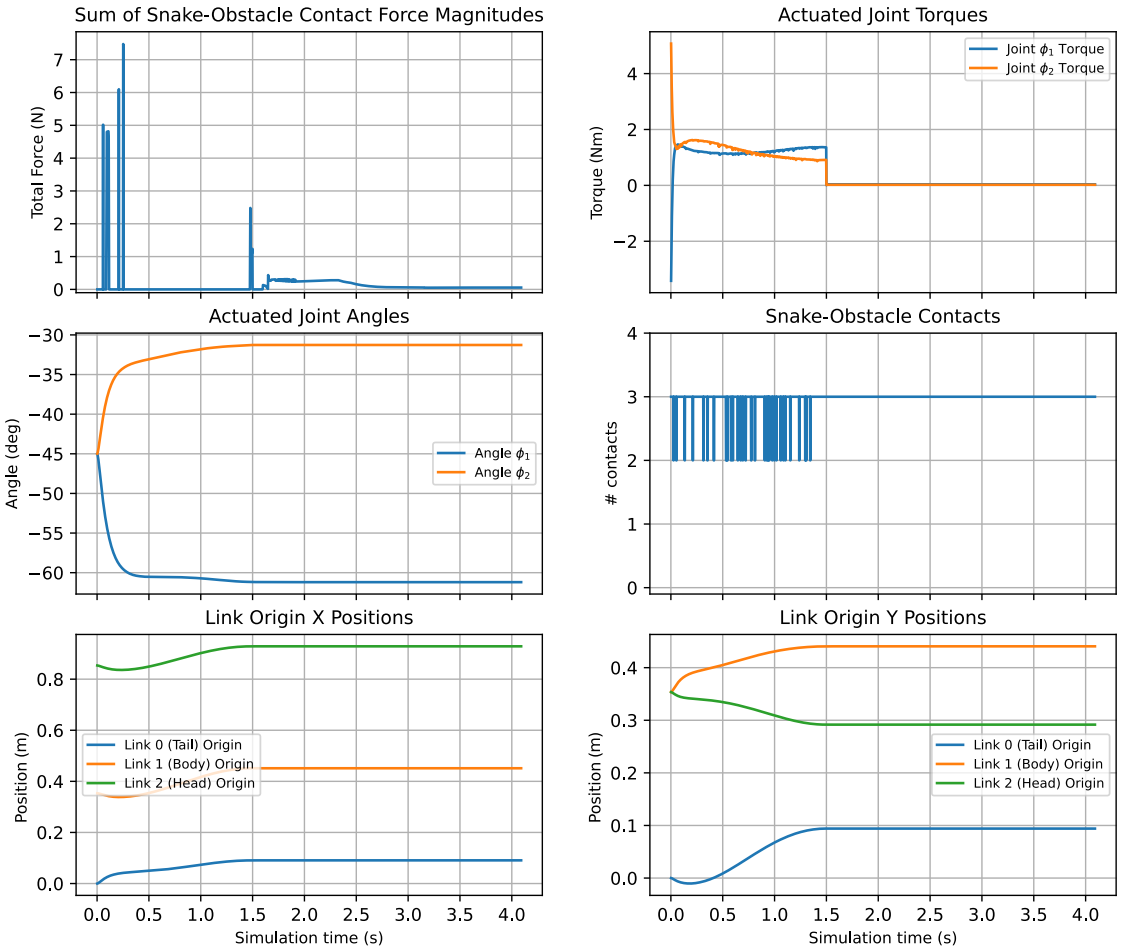


Figure 5.4: Experimental results for achieving and holding a target non-symmetric static pose.

and maneuvers the snake into form closure. The crucial difference is that the contact force is reduced, and due to redundancy, is not presented in the following figures. Phase B then initiates a rocking movement with active force control. The force reference is chosen to be much smaller (0.5 N) to engage the full range of motion. The specific rocking motion parameters used for the reference $\theta_{\text{ref},i}$ of Equation (4.1) are: center angle $\theta_c = -0.7854\,\text{rad}$ ($\approx -45°$) for each joint, amplitude $A_\theta = 0.5236\,\text{rad}$ ($\approx 30°$), temporal frequency $\omega_t = 1.0\,\frac{\text{rad}}{\text{s}}$, and spatial phase difference $\Delta\phi_s = \pi\,\text{rad}$, approximating the values given in Section 4.1.2.

5.2.1 Rocking Motion with Stepped Force Activation

The goal of this experiment is to observe the system response to dynamically changing target force levels during a rocking motion. The force activation level k is increased in a stepped profile: $k = [0.5 \rightarrow 1.0 \rightarrow 1.5 \rightarrow 2.0 \rightarrow 2.5]$, with each level maintained for a duration of 2 seconds during phase B. Figure 5.5 shows a series of images of the simulated snake robot performing the rocking motion under the influence of the k-profile.

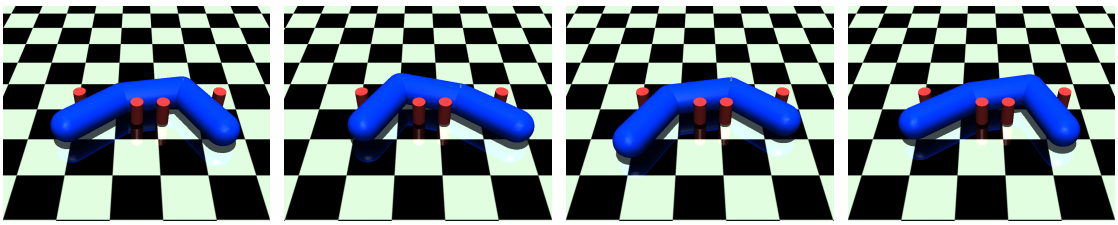


Figure 5.5: Visualization of a snake robot performing a rocking motion with a stepped k-profile for force activation.

Figure 5.6 presents the quantitative data recorded during the experiment. The "Actuated Joint Angles" and "Link Origin X/Y Positions" exhibit periodic variation, with the individual joints showing phase-shifted sinusoidal patterns. The "Sum of Snake-Obstacle Contact Force Magnitudes" displays groups of high-magnitude spikes that alternate with intervals where the forces are close to zero. These spike groups increase in average magnitude as the k level increases. A similar pattern is observed in the "Snake-Obstacle Contacts", where contact events vary rapidly between two and three across time. The "Actuated Joint Torques" exhibit similar spiking to the force magnitudes. A periodic pattern can be identified within the noise.

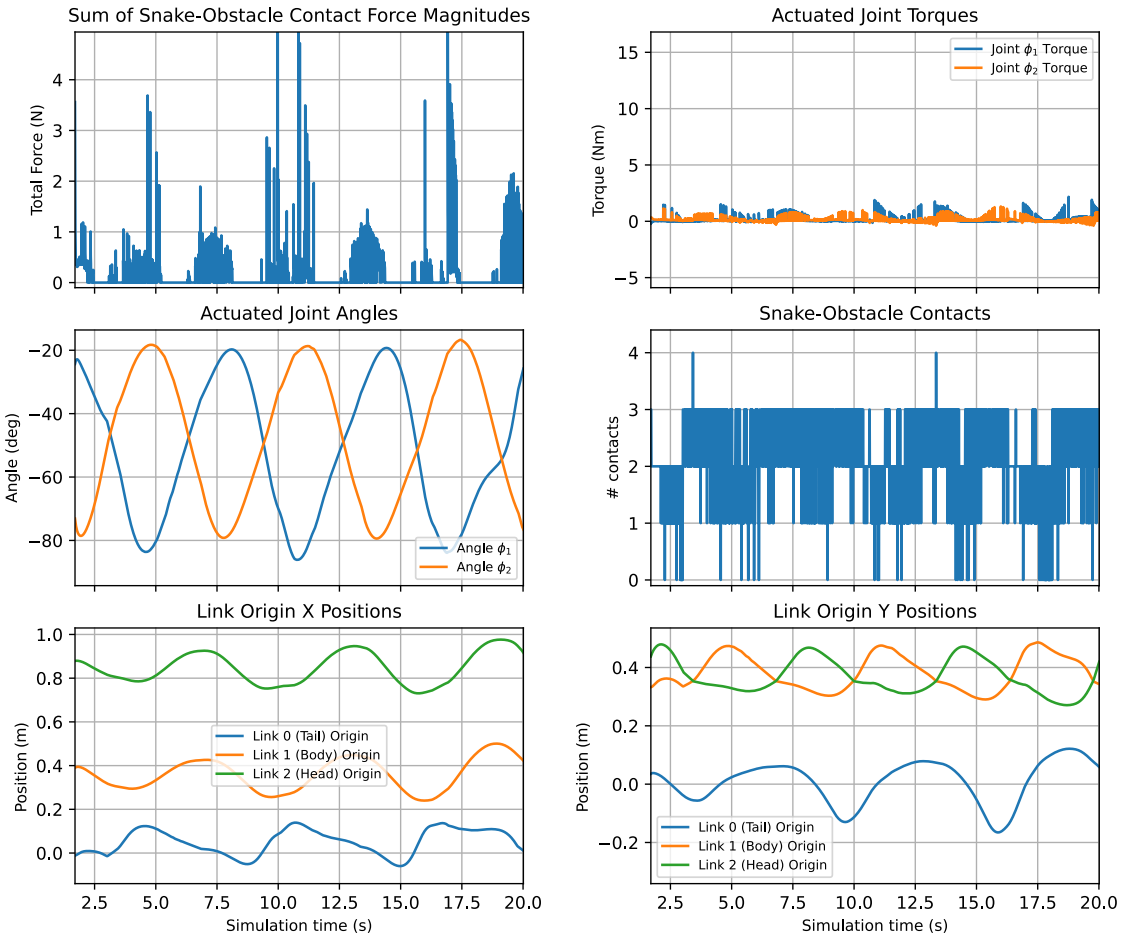


Figure 5.6: Experimental results for rocking motion with a stepped k-profile.

5.2.2 Rising Force Activation

This experiment investigates the system behavior under a continuously increasing force activation level during the rocking motion, with k increasing linearly from 0.5 to 100. Figure 5.7 presents the results. Initially, the plots exhibit similar characteristics to those observed in the stepped k-profile experiment. Over time, however, the motion pattern begins to change. Around 18 seconds into the simulation, the "Link Origin X/Y Positions" show a noticeable reduction in variation, and the rate of position change diminishes. Concurrently, the "Snake-Obstacle Contacts" plot shows an increase in the average number of simultaneous contacts, rising to approximately four. The "Actuated Joint Angles" and "Link Origin X/Y Positions" show reduced amplitude over time after 18 seconds, eventually flattening out. No constant "Sum of Snake-Obstacle Contact Force Magnitudes" or "Actuated Joint Torques" can be observed after the motion comes to a halt.

5.2.3 Long Time Simulation of Sustained Rocking Motion with Constant Force Activation

This experiment investigates the behavior of the system during an extended simulation time of 60 seconds with a constant force activation level, which set to $k = 1.0$ throughout phase B. Figure 5.8 presents the recorded data over a prolonged time period. The "Actuated Joint Angles" and "Link Origin X/Y Positions" exhibit consistent periodic oscillations without noticeable drift. The "Sum of Snake-Obstacle Contact Force Magnitudes" and "Actuated Joint Torques" continue to display sharp transient peaks, with values fluctuating around a relatively constant average. The "Sum of Snake-Obstacle Contact Force Magnitudes", "Actuated Joint Torques", and "Snake-Obstacle Contacts" share oscillatory patterns.

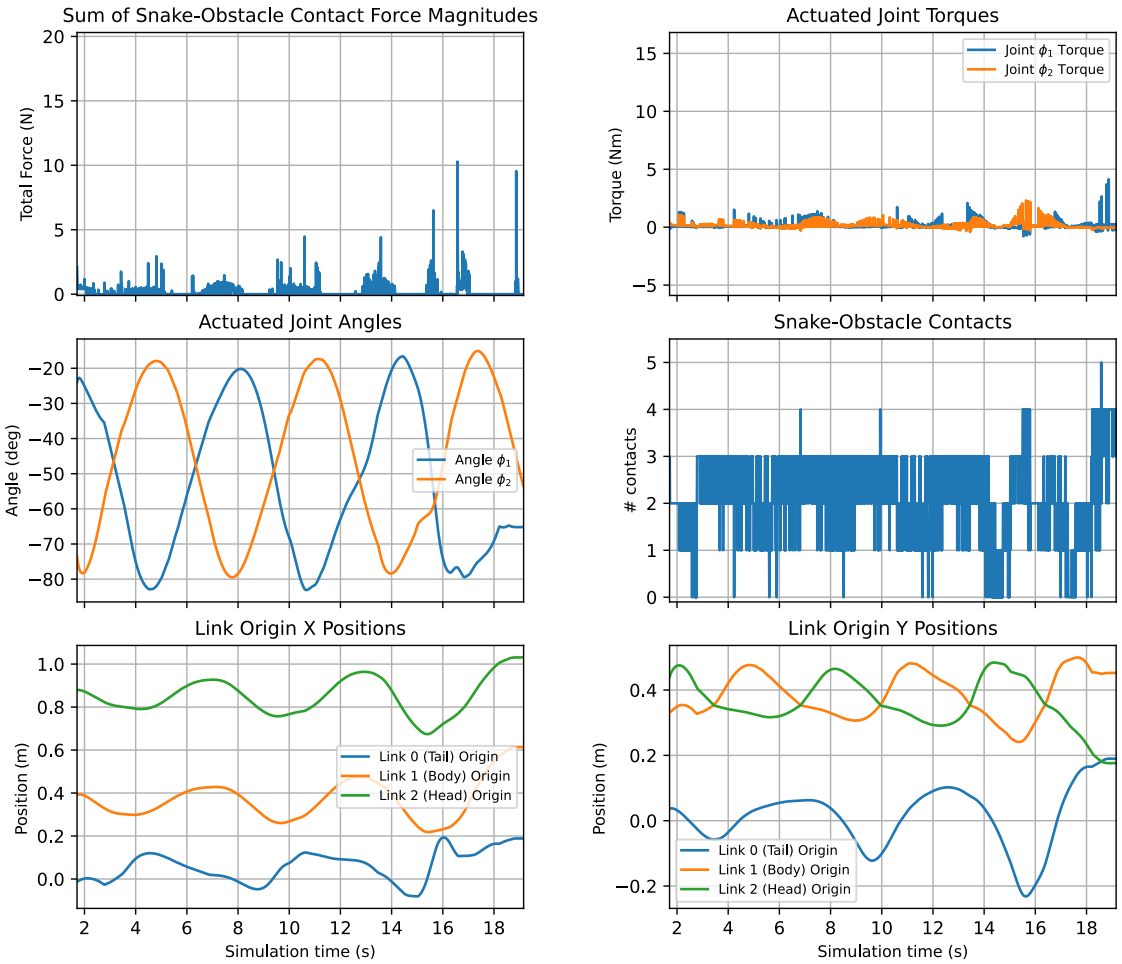


Figure 5.7: Experimental results for rocking motion with a rising k-profile.

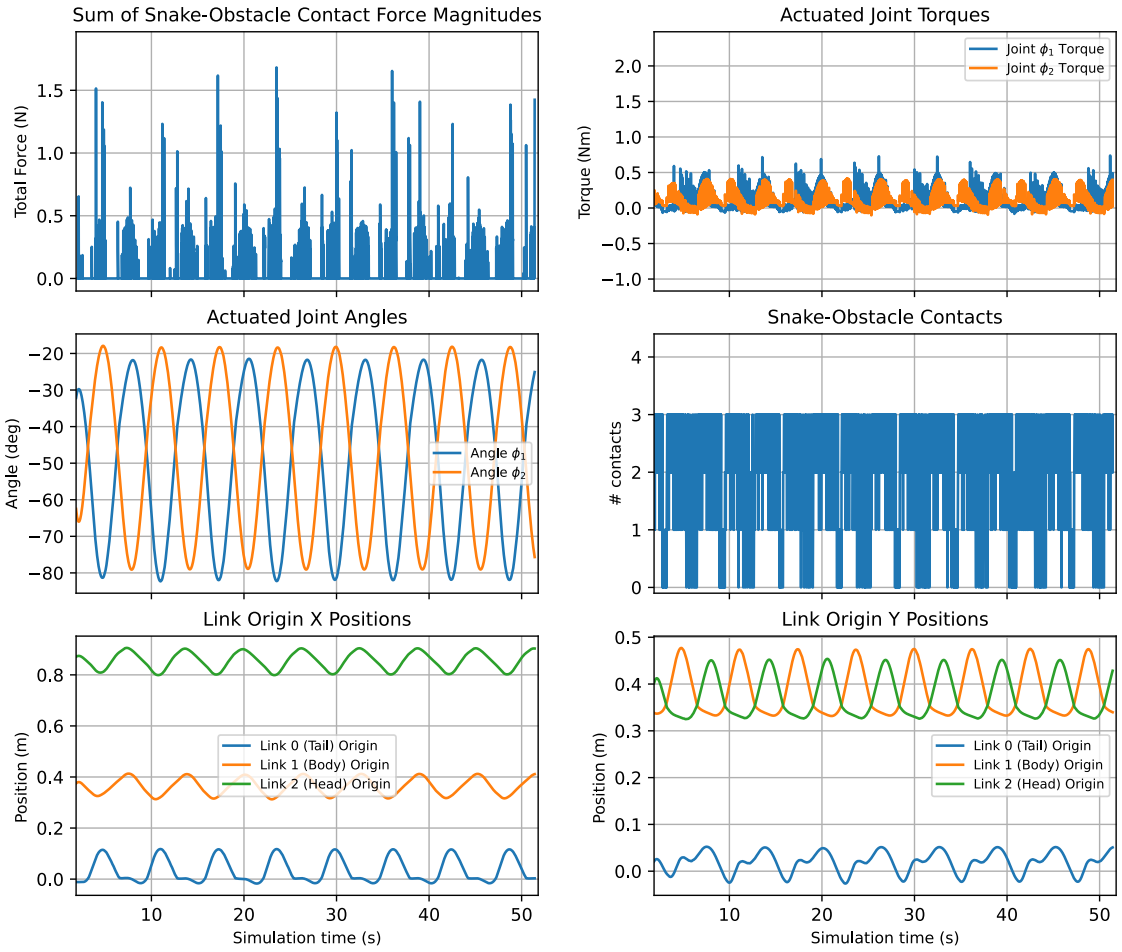


Figure 5.8: Experimental results for sustained rocking motion with a constant force activation level with constant periodic pose oscillations.

6 Discussion

This study has demonstrated that Hybrid Position and Force Control, or HPFC for short, when combined with Obstacle-Aided Locomotion, or OAL for short, presents a highly complex control challenge. Designing appropriate trajectories and computing the corresponding task and manipulator Jacobians proved to be non-trivial, even for the simple example presented in Chapter 4. Executing Hybrid Obstacle-Aided Locomotion, or HOAL for short, demands careful coordination and remains a challenging task, even under simplified conditions. Initial efforts extended the proposed method to more complex trajectories. However, due to a combination of factors, such as unpredictable simulation errors and scope limitations, these extensions were ultimately excluded from the main evaluation.

Section 6.2 provides an overview of the trajectory generation process developed for these more advanced configurations and outlines directions for future work. The corresponding implementations have already been developed as modular, reusable code. Section 6.3 discusses the development process of the simulator itself and gives an overview of the current state of *SimSerpent*. This section highlights issues that arose during the creation process, as well as the found solutions. Beyond that, unresolved problems and evaluations are addressed. The following section reviews the results found in Chapter 5 and evaluates the initial research question of the thesis.

6.1 Discussion of Results

Some results have already been presented in Section 4.1 for illustrative purposes. Figure 4.4 presents one of the first results of the thesis, showing that an arbitrary configuration of a snake robot can be put into the simulation. In the simplified 3-link configuration, the

bodies remain motionless, proving initial stability of the simulated system. Figure 6.3 is created using the actual output of the simulation and reflects the expected behavior. This shows that the movements follow the trajectory of Equation (4.1) when minuscule friction and no obstacles are present.

Symmetric Static Pose Scenario: The visualizations in Figure 5.1 indicate that the contact force vectors align as expected, both in magnitude and orientation. This suggests correct implementation of the contact force control within the form closure configuration.

The data in Figure 5.2 supports the conclusion that the two-phase control strategy does not induce instability, since all the target values are met and stay at a constant level. Despite a discrete and instantaneous change in the target contact force, the "Actuated Joint Torques" adapt accordingly, while the "Actuated Joint Angles" and "Link Origin Positions" remain consistent after the change. There are four "Snake-Obstacle Contacts" during the entire simulation time, which is compatible with form closure. This behavior indicates that the system maintains its static pose and that force control is effectively decoupled from position control under static conditions. Moreover, the two-phase approach appears to facilitate form closure, and thereby can be used for an effective system initialization without observable drawbacks.

Target Asymmetric Static Pose Scenario: Figure 5.3 shows that the snake reaches and maintains the target non-symmetric pose with a slight offset, which is held after motion control is disabled in phase B. The corresponding plot in Figure 5.4 supports this observation, as both "Actuated Joint Angles" and "Link Origin Positions" remain at the commanded levels during phase B. This confirms the basic ability of the controller to achieve and hold non-symmetric static configurations under low force requirements. However, the "Sum of Snake-Obstacle Contact Force Magnitudes" exhibits significant spiking during the beginning of phase A, particularly while contact forces are applied and movement happens simultaneously. This behavior is likely caused by transient vibrations leading to brief contact disruptions. Despite extensive debugging and parameter tuning, including adjustments to friction, damping, and control gains, this issue persisted, suggesting that it originates from limitations in the simulation environment and its force sensing capabilities, rather than the controller itself.

What stands out is the permanent violation of form closure, seen in the plot of "Snake-Obstacle Contacts". Although a maximum of three contact points can be seen, the snake robot does not leave its position and is able to maintain it. In theory, this should not be possible with only three contact points, which could indicate an additional factor of damping. One explanation may be that even with a small penetration depth, minuscule friction has a greater effect than initially assumed. It is also conceivable that this is connected to the problem of transient vibrations. There might exist a fourth contact point which is not recognized by the simulation. This problematic behavior continues for the remainder of the test results and is thereby not mentioned in further analysis. For the following tests, the results of the initial phase A remain the same as shown in the first test *Symmetric Static Pose Scenario* and are not included in further discussions to avoid repetitiveness.

Rocking Motion with Stepped Force Activation: Due to the spiking artifacts in the sensor data, particularly in the "Sum of Snake-Obstacle Contact Force Magnitudes" plot in Figure 5.6, only qualitative trends can be assessed. As in the asymmetric test scenario, the contact force values fluctuate between periods of zero and high-amplitude spikes. However, when considering the average level within each spike group, an increasing trend is recognizable in correspondence with the stepped increase in the parameter k. This suggests that the rising k-profile contributes to an increasing overall contact force, despite the presence of corrupted measurements.

The "Snake-Obstacle Contacts" values are similarly affected, but frequently maintain a value of three, indicating that form closure is not preserved. While the contact related plots exhibit noise, the "Actuated Joint Angles" and "Link Origin X/Y Positions" demonstrate periodic behavior consistent with the expected rocking motion. The joint angle curves resemble those of the reference motion shown in Figure 4.4, particularly during the lower k-levels. As the k-levels rise, the motion becomes increasingly distorted but still recognizable. Together, these observations indicate that the controller preserves a stable motion pattern under stepped force modulation, even in the presence of sensor artifacts and the absence of form closure. Given that real-world force measurements often contain inherent noise and movement always induces vibrations, the minimal impact observed on the velocity profile highlights a degree of practical robustness in the proposed HPFC controller.

Rising Force Activation: As in the previous scenario, the system initially exhibits periodic motion with phase-shifted joint oscillations. However, as the force activation level k increases, the influence of the force control component becomes more pronounced, but movement continues to function as intended. This leads to a gradual amplification and overshooting of the rocking motion, since the motion control has to fight the force control in order to maintain the desired movement. In this test, around the 18-second mark, the "Actuated Joint Angles" and "Link Origin X/Y Positions" begin to flatten, indicating a transition to a near-static configuration, where the motion control can no longer overpower the force control. The "Snake-Obstacle Contacts" simultaneously increase in average number, showing that the snake is pressing more firmly into its environment.

This behavior aligns with the expectations: as the commanded contact force increases, the force control component dominates, eventually overriding the periodic motion input. The resulting state, where motion ceases and the snake maintains obstacle contact, demonstrates a limit case of the HPFC in which force control is reinforced at the cost of mobility.

Long Time Simulation of Sustained Rocking Motion with Constant Force: This final experiment is designed to evaluate the consistency of the observed oscillatory behavior under a constant force input over an extended time duration. The results in Figure 5.8 demonstrate that the system maintains periodic motion across all relevant variables while the force activation level remains constant.

The "Actuated Joint Angles" and "Link Origin X/Y Positions" display repeatable oscillatory patterns without degeneration or drift, suggesting that the HPFC controller can preserve repeating motions over time. Despite the presence of spikes in the "Sum of Snake-Obstacle Contact Force Magnitudes" and "Actuated Joint Torques," the overall periodic motion pattern remains consistent. The disturbances may reflect realistic, rapid variations in force measurements, rather than indicating flaws or numerical artifacts in the HPFC implementation. However, it must be acknowledged that a definitive conclusion cannot be drawn. Given that velocities are integrals of forces and inherently filter rapid fluctuations, noisy force signals are plausible and expected in practical scenarios. These findings support the conclusion that the HPFC maintains resilient behavior during continuous dynamic tasks and that previously observed spiking does not disrupt the overall system performance.

Summary: The experimental results demonstrate that the kinematic HPFC approach is capable of maintaining static poses and executing periodic motions under varying force conditions, even in the presence of unknown simulation behavior. The controller successfully decouples position and force objectives during static tasks and shows stable, repeatable behavior under both stepped and continuously increasing force inputs. While certain limitations related to simulation accuracy persist, they do not compromise the overall reliability and effectiveness of the control strategy.

These findings validate the design of the HPFC implementation and support its suitability for further development and more complex scenarios. Building on these results, the following Section 6.2 outlines potential improvements, extensions, and research directions aimed at advancing the controller capabilities and addressing further challenges.

6.2 Outlook

While the experiments in Chapter 5 confirm the basic functionality and resilience of the HPFC in a simplified testbed, the broader goal remains to apply the controller in more complex, or even unpredictable environments. This chapter outlines several directions for future research and system development. These include generating parameterized trajectories for long-term stability testing, developing strategies for physically consistent snake initialization, devising methods for evaluating tracking performance, and creating a catalog of test scenarios designed to demonstrate specific controller capabilities. Together, these components provide a foundation for advancing the *SimSerpent* framework toward more realistic and demanding applications. The following section outlines an approach for generating complex trajectories, which extends the subject of paths addressed in Section 3.3.1.

6.2.1 Trajectory Generation

This section presents the motivation for selecting a specific trajectory to evaluate the reliability of the controller. The trajectories discussed in this section follow the conventions of a *path* that are stated in Section 3.3.1. The objective is to ensure the long-term

repeatability of the control behavior within a constrained trajectory framework in which the snake robot advances along the path.

Two general strategies were considered for testing long-term path following. The first strategy involves a long, linear trajectory spanning several times the length of the snake to demonstrating its ability to follow a path over extended distances. The second approach relies on a circular trajectory, to enable cyclic motion and repeated interaction with the environment.

Long Linear Trajectory The long linear trajectory offers the advantage of simplicity and ease of implementation. It can be constructed by repeating the pattern introduced in Chapter 4. However, it has computational drawbacks. Despite its repetitive nature, the entire trajectory must be loaded and maintained in the simulation, which may not be memory efficient. These limitations could potentially be addressed through more advanced simulation techniques or trajectory handling methods.

In contrast, a circular trajectory naturally allows for repeatability without requiring additional memory resources. This makes it ideal for continuous simulations in which the snake repeatedly engages with predefined contact points. However, the drawback of this approach lies in its geometric complexity, which is discussed in the following section.

Circular Trajectory Generation This section outlines the procedure for constructing a closed, obstacle-constrained trajectory using geometric primitives. The resulting path is designed to be continuously traversable by the simulated snake robot while maintaining form closure and geometric smoothness. The process is illustrated in Figure 6.1, which depicts the nine key steps involved in path construction and obstacle placement.

The generation process begins with a predefined radius r_p, which is used to create the main circle ("Step 1"). To ensure a closed, symmetric shape, the circle is divided into $2n_\mathrm{p}$ equal segments ("Step 2"). In the illustrated example, $n_\mathrm{p} = 6$, yielding 12 evenly spaced intersections. From these intersections, alternating inclined and declined construction lines are then drawn at angles of $\alpha_\mathrm{p} = \pm 40°$ ("Step 3"). Next, line segments of length $m = 0.2$ are drawn from each intersection point in both directions along these

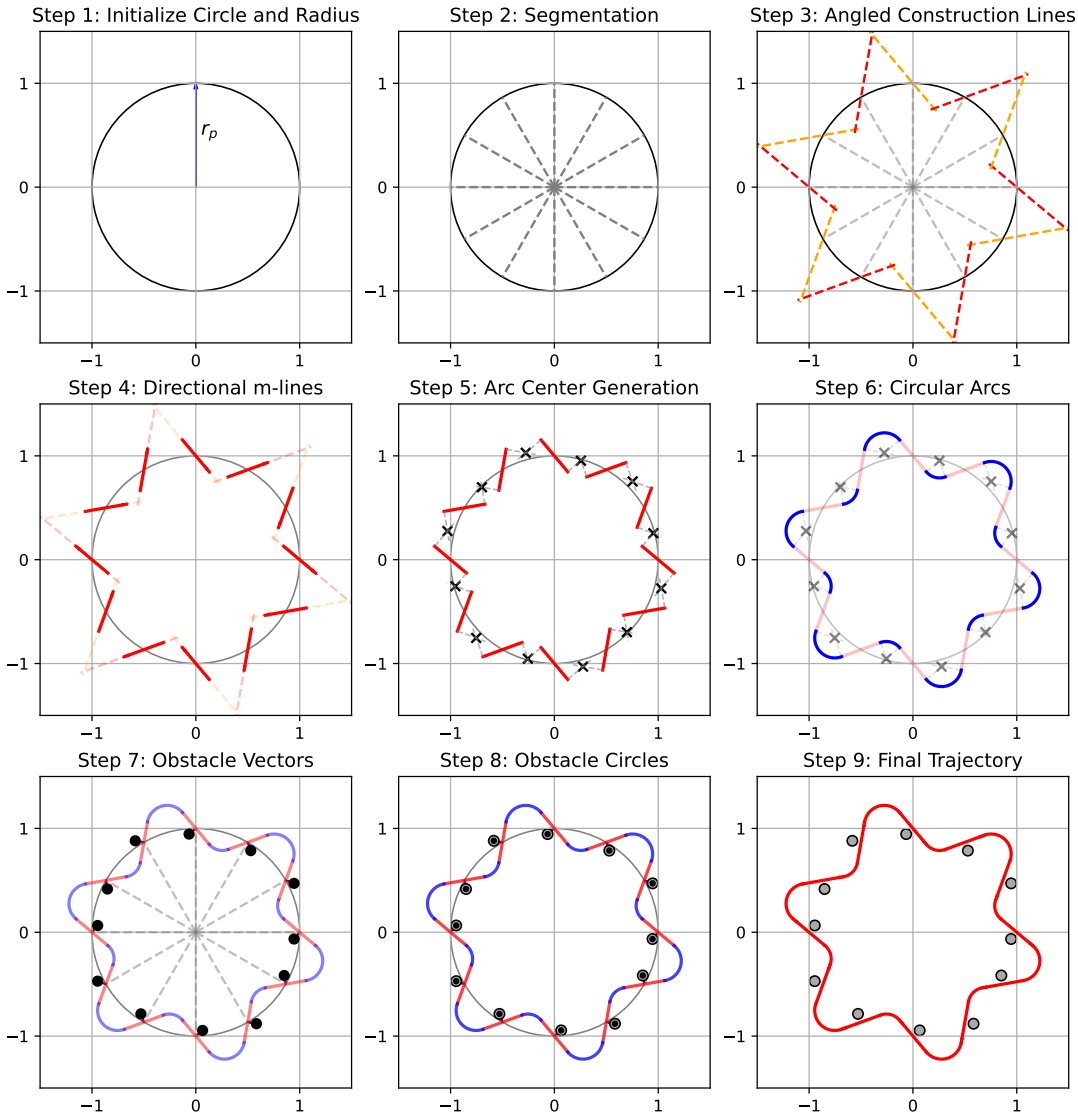


Figure 6.1: Step-by-step generation of a circular trajectory with obstacles.

construction lines ("Step 4"). These segments are referred to as m-lines and form the basis for the polygonal geometry.

To ensure geometric consistency, a triangle-based construction is used to derive a constraint for the maximum permissible value of m in relation to r_p. Consider a triangle where the base is r_p that connects the origin $A = [0, 0]$ to point $B = [0, r_\mathrm{p}]$. The half-sector angle is $\alpha_\mathrm{A} = \frac{\pi}{2n_\mathrm{p}}$, and the attack angle at B is $\alpha_B = \alpha_\mathrm{p}$. Let a be the side opposite α_p, which connects point C to A. Applying the law of sines yields:

$$\frac{a}{\sin(\alpha_\mathrm{p})} = \frac{r_\mathrm{p}}{\sin\left(\alpha_\mathrm{p} + \dfrac{\pi}{2n_\mathrm{p}}\right)}.$$

The side length a can be computed as

$$a = \frac{r_\mathrm{p} \cdot \sin(\alpha_\mathrm{p})}{\sin\left(\alpha_\mathrm{p} + \dfrac{\pi}{2n_\mathrm{p}}\right)},$$

yielding the geometric bound $m < a$.

With this in mind, a direction vector of length $2m$ is then assigned to each m-line. The parameter m is chosen based on the physical dimensions of the snake robot (*Boa*), as listed in Table 2.1, and r_p is tuned accordingly.

In "Step 5", perpendicular lines are constructed at the endpoints of the m-lines, and their pairwise intersections are computed. It is important to select the correct intersections according to line orientation. These intersections define the arc centers used in "Step 6", where smooth, blue connecting arcs are drawn to join the endpoints. Each arc is defined by the distance between an endpoint and its corresponding intersection point, as well as the calculated intersection angle. This ensures geometric continuity and prevents kinks in the trajectory.

"Steps 1–6" define the closed trajectory. To enable propulsion while maintaining constant form closure potential, obstacles must be added. In "Step 7", orthogonal vectors of length $\varepsilon_\mathrm{c} + r_\mathrm{c}$ are placed at alternating segmentation points along the main circle. Since these vectors are relatively short, "Steps 7 and 8" are shown in more detail in Figure 6.2.

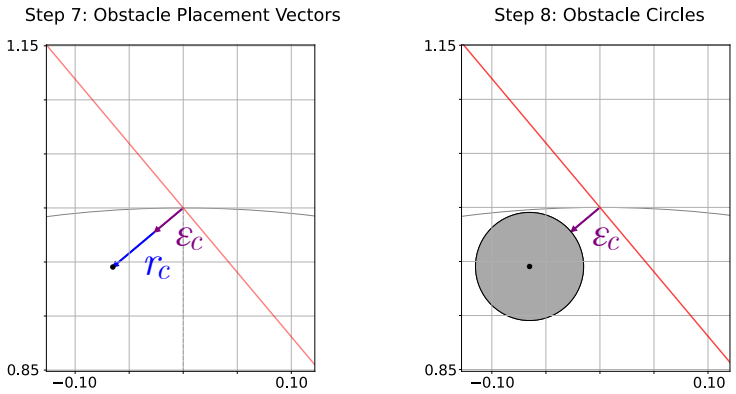


Figure 6.2: Zoomed-in view of Steps 7 and 8 for obstacle placement.

Here, ε_c represents a safety margin related to the robot's thickness, and r_c is the radius of each obstacle. To ensure point contacts, r_c should be chosen no greater than $2\varepsilon_c$. In "Step 9", all obstacles are placed (shown in gray), completing the construction.

Although Figure 6.1 illustrates the default configuration, the implemented code is fully modular and supports alternative settings. Variants are presented in Figure 6.3, where the obstacle distribution may differ.

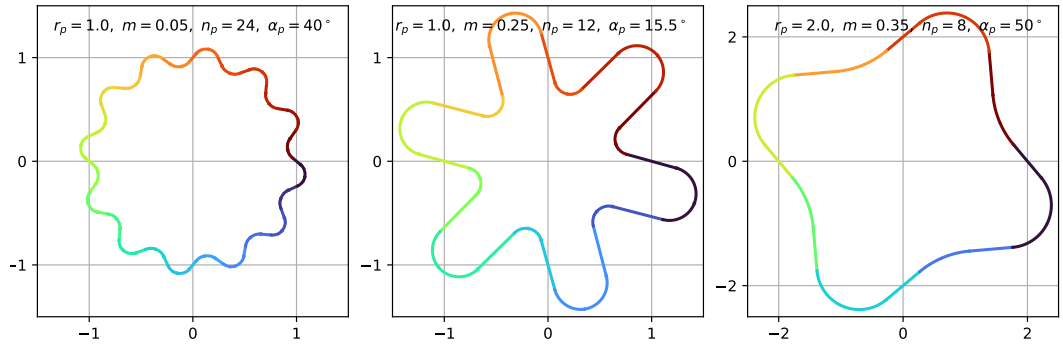


Figure 6.3: Alternative settings for circular trajectory generation.

As illustrated in Figure 6.1, this layout has proven effective for snake robots with 20 links. The resulting trajectories are compact and continuous, making them suitable for repeated simulation. They satisfies the geometric and physical requirements for HOAL and serve as a reference path for the initialization and evaluation of the snake controller, as discussed in the following Section 6.2.2.

6.2.2 Snake Initialization for Complex Trajectories

This section describes how the snake robot is placed on a circular trajectory at the start of the simulation. The goal is to establish a physically consistent, form closed, initial configuration with zero joint error, serving as the basis for subsequent controller evaluation.

Parameterized Curve Definition: As constructed in Section 6.2.1, the closed trajectory consists of segments, alternating between straight m-lines and circular arcs. A global curve parameter $\alpha \in [0, 1)$ is introduced to evaluate positions along the trajectory:

$$\alpha_{\text{big}} = \alpha \cdot 2n_{\text{p}}, \quad i = \lfloor \alpha_{\text{big}} \rfloor, \quad t = \alpha_{\text{big}} - i.$$

The position $\mathbf{p}(\alpha) \in \mathbb{R}^2$ is determined based on whether segment i is linear or curved. If i is even (straight m-line segment):

$$\mathbf{p}(\alpha) = (1 - t)\, \mathbf{E}_{\text{start}}^{(i/2)} + t\, \mathbf{E}_{\text{end}}^{(i/2)}.$$

If i is odd (circular arc): Let $j = \lfloor i/2 \rfloor$, and define endpoints $\mathbf{p}_1, \mathbf{p}_2$ as:

$$\mathbf{p}_1 = \begin{cases} \mathbf{E}_{\text{start}}^{(j)}, & \text{if } (j+1) \bmod 2 = 0 \\ \mathbf{E}_{\text{end}}^{(j)}, & \text{otherwise} \end{cases}, \quad \mathbf{p}_2 = \begin{cases} \mathbf{E}_{\text{start}}^{((j+1) \bmod n_{\text{p}})}, & \text{if } (j+1) \bmod 2 = 0 \\ \mathbf{E}_{\text{end}}^{((j+1) \bmod n_{\text{p}})}, & \text{otherwise} \end{cases}.$$

Using the arc centers $\mathbf{c}^{(j)}$ and radii $r^{(j)}$, compute:

$$\alpha_1 = \operatorname{atan2}((\mathbf{p}_1 - \mathbf{c}^{(j)})_y,\ (\mathbf{p}_1 - \mathbf{c}^{(j)})_x), \quad \alpha_2 = \operatorname{atan2}((\mathbf{p}_2 - \mathbf{c}^{(j)})_y,\ (\mathbf{p}_2 - \mathbf{c}^{(j)})_x)$$

$$\Delta\alpha = ((\alpha_2 - \alpha_1 + \pi) \bmod 2\pi) - \pi, \quad \alpha_{\text{arc}} = \alpha_1 + t \cdot \Delta\alpha$$

$$\mathbf{p}(\alpha) = \mathbf{c}^{(j)} + r^{(j)} \begin{bmatrix} \cos(\alpha_{\text{arc}}) \\ \sin(\alpha_{\text{arc}}) \end{bmatrix}$$

Joint Placement Along the Trajectory: After parameterizing the trajectory, the snake can be initialized in a consistent configuration. Each joint must lie on the trajectory while maintaining the physical constraint of constant segment length. The initialization begins at a fixed tail position, chosen at $[0,\ r_{\text{p}}]$ or $\alpha_{\text{big}} = 0$ (see Figure 6.4). A circle with

a radius of m is drawn around this point to identify the feasible region for the next joint. The intersection of this circle with the trajectory is computed, and only the intersection lying in the clockwise direction is retained. The placement procedure is geometric by nature, and one representative step is shown in Figure 6.4. This process is repeated for each new joint position: a circle of radius m is drawn and its intersection with the trajectory is determined. The corresponding segment vector is stored, and the iteration continues until the desired number of joints is placed (e.g., $n = 20$) is placed, as shown in the last subfigure of Figure 6.4. From a programming perspective, parameterizing the curve is somewhat involved. However, it enables the fast, accurate identification of intersection points using an optimization-based approach.

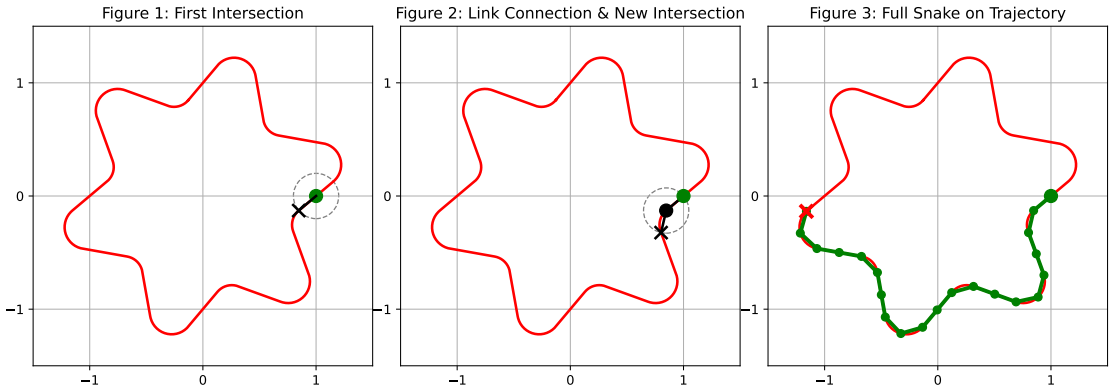


Figure 6.4: Illustration of a step-by-step snake initialization process.

Joint Angles and Special Cases: During initialization, the relative joint angles, denoted by ϕ_i, are computed as illustrated in Figure 3.4. Particular attention is paid to joints positioned at contact points with obstacles, which coincide with segmentation points (see "Step 2" in Figure 6.1). These joints are checked for near zero relative angles, which reflect the expected straight alignment at obstacle contacts. Due to numerical limitations, exact zero values cannot be guaranteed; however, deviations are typically negligible. This initialization method provides a starting configuration with zero error from a numerical point of view and supports consistent evaluation.

6.2.3 Error Metrics for Tracking Evaluation

Two distinct types of error are considered to evaluate the performance of the controller: the geometric deviation of the snake's head or tail from the reference path and the configurational deviation of the overall body shape from the desired configuration.

1. *Geometric Error (Head-to-Path Distance):* Spatial deviation is measured by computing the distance from the head of the snake to the reference trajectory in an orthogonal direction. This provides a simple indication of how well the leading segment is following the intended path. It is also used to detect significant positional drift. The curve is parametrized to efficiently project the head's position onto the path. The shortest distance is calculated using the numerical optimization-based routine mentioned in the previous section.
2. *Configurational Error (Form Error via Joint Angles):* Although the head position offers a general indication of spatial accuracy, the overall shape of the snake is better described by the joint angles, which represent the internal articulation between adjacent segments and are directly controlled by the system.
   To quantify configurational accuracy, the current joint angles $\phi(t_i)$ are compared against a reference configuration $\phi_{\text{ref}}(t_{i+1})$, which corresponds to the ideal placement of the snake on the trajectory at the next expected position:

   $$e_\phi(t_i) = \phi(t_i) - \phi_{\text{ref}}(t_{i+1})$$

   The reference configuration can be obtained through online computation or by using a precomputed lookup table. Although lookup tables can reduce runtime in long simulations, they are not necessary for this implementation. The optimization-based intersection finding algorithm used here, enables fast and accurate online computation, rendering precomputation optional.

Assumptions and Simplifications: Because of the continuous form closure assumption, the snake is expected to stay close to the path throughout the simulation. Consequently, geometric error is usually minimal, and configurational error ($e_\phi(t_i)$) is the main metric for evaluating the controller. This focus is justified by the control architecture, which operates at the joint level and cannot directly command Cartesian positions.

6.2.4 Design of Future Test Scenarios

An extended set of test scenarios was developed to explore how the HPFC might perform under a range of conditions. These scenarios are included here to provide a structured basis for future experiments, although they are not evaluated in this thesis.

Each test highlights specific aspects of the controller behavior, such as its sensitivity to sensor disturbances, initialization offsets, and variations in trajectory geometry. The scenarios also guided the development of the simulation environment, which was expanded to include features such as noise injection and flexible speed profiles. Table 6.1 summarizes the core parameters of each test case, and detailed descriptions follow.

Table 6.1: Overview of extended test scenarios to validate the HOAL approach.

No.	Trajectory Parameters	Sen. Noise	Init. Err.	Obs. Off.	Speed
1	$r_\mathrm{p} = 1.0$, $\alpha_\mathrm{p} = 40°$, $n_\mathrm{p} = 12$	No	No	No	Slow
2	$r_\mathrm{p} = 1.0$, $\alpha_\mathrm{p} = 40°$, $n_\mathrm{p} = 12$	No	No	No	Fast
3	$r_\mathrm{p} = 1.0$, $\alpha_\mathrm{p} = 40°$, $n_\mathrm{p} = 12$	Yes	Yes	No	Slow
4	$r_\mathrm{p} = 1.0$, $\alpha_\mathrm{p} = 40°$, $n_\mathrm{p} = 12$	No	Yes	Yes	Slow
5	$r_\mathrm{p} = 5.0$, $\alpha_\mathrm{p} = 40°$, $n_\mathrm{p} = 24$	No	No	No	Slow

Test 1: *Ideal Baseline:* A moderately curved trajectory is used, with no sensor noise, initialization error, or obstacle offset. The snake moves slowly. This setup serves as a baseline to verify the fundamental functionality of the controller under idealized conditions.

Test 2: *Increased Actuation Speed:* This scenario is identical to Test 1, but with a higher movement speed. This scenario can be used to examine how the controller handles increased dynamics and determine if tracking accuracy deteriorates with faster actuation.

Test 3: *Sensor Noise and Misalignment:* Additive white noise (10% of the signal amplitude) is introduced to the sensor readings. Additionally, the snake is initialized with a small offset. This test examines how sensitive the controller is to imperfect sensing and positioning.

Test 4: *Environmental Uncertainty:* In this test the sensor data is assumed to be accurate but the physical obstacles are slightly offset from their expected locations. Consequently, the snake is initialized with a misalignment. The controller continues to rely on the unmodified contact map. This test investigates the sensitivity of the system to mismatches between the environment model and reality.

Test 5: *Low-Curvature Trajectory:* Using a larger trajectory radius increases the total path length and reduces the local curvature. This changes the frequency and nature of contact events and reveals how effectively the controller performs in a geometry closer to a straight path.

Together with the modular implementation described in this chapter, these scenarios form a ready-to-use framework for exploring controller performance under increasingly realistic conditions. The proposed test scenarios are supported by a modular implementation that enables flexible experimentation. However, a number of unexpected challenges emerged during the integration of the HPFC and trajectory logic into the simulation environment. Though secondary to the primary evaluation, these technical difficulties provide valuable insight into the limitations of the simulator and the practical constraints of the setup. The following section documents these implementation specific observations and outlines realizations made during debugging and refinement.

6.3 Implementation Challenges and Technical Observations

Integrating trajectory generation, control algorithms and initialization routines into *SimSerpent* revealed several subtle yet significant issues related to the initially provided simulation setup, numerical stability, and visual alignment within *SimSerpent*. This section summarizes the key challenges and highlights potential pitfalls for future extensions.

Trajectory Placement Challenges: Accurate trajectory placement required determining joint positions along trajectory segments while maintaining constant link lengths. Initially, analytical methods based on circle-circle intersections were employed, but they frequently failed due to ambiguous intersection cases at the transitions between segments. These issues were resolved by shifting to a parameterized trajectory representation, as

explained in Section 6.2.2, and computing joint positions numerically at evenly spaced lengths along the curve. However, solver convergence deteriorated for snake configurations exceeding 18 links. This issue was effectively addressed by relaxing the numerical solver tolerance from 1×10^{-16} to 1×10^{-10}.

Although the initial implementation of numerically sampling the trajectory successfully placed joints along it, frequent ambiguous intersections and numerical convergence challenges emerged. This motivated the shift toward the parameterized curve formulation, which proved to be computationally efficient and accurate.

Visual Misalignment at Initialization: Despite the low tolerance for joint placement, initial visual misalignment occurred, manifesting as abrupt positional adjustments at the onset of the simulation. Investigated hypotheses included geom-body offsets, joint angle sign inversions, directionality (head-first vs. tail-first), and relative yaw computations. However, none of these hypotheses fully resolved the issue. Eventually, consistency in configuration parameters, particularly segment lengths, and correcting orientation offsets—notably removing an unintended $\frac{\pi}{2}$ offset and applying a π rotation correction to the head quaternion—significantly improved alignment.

Due to residual controller logic, particularly from an old test scenario with sine-based excitation modes, significant unintended joint forces appeared immediately upon initialization. Disabling this mode (`do_sinus = False`) did not resolve immediate issues. Further analysis revealed that even passive tests (with control callbacks disabled) exhibited unwanted straightening. This was caused by non-zero position actuator gains (`position_actuator_KP`), which introduced a bias, spring-like force toward zero joint angles. Disabling the direct setting did not solve this problem. Only setting the actuator gains to zero effectively eliminated this issue. The presence of legacy controller routines indicated incomplete modularity within *SimSerpent*, which motivated the simplification and restructuring of the simulation environment.

Nevertheless, it was possible to implement the snake in this complex environment, as can be seen in Figure 6.5. The red cylinders represent obstacles, and the small blue spheres represent points on the circular trajectory.

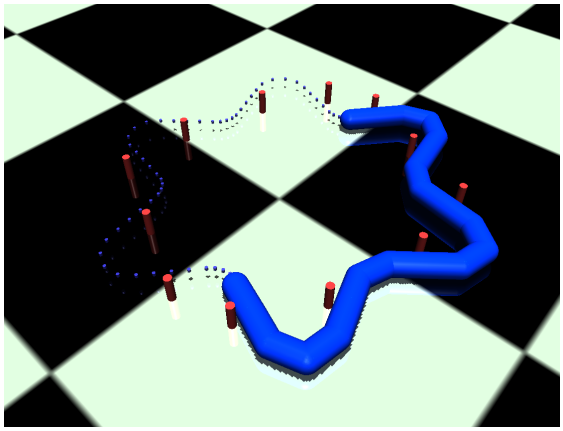


Figure 6.5: Simulated snake robot spawned on a circular trajectory.

6.3.1 Motivation for the Simplified 3-Link Model

The original simulator occasionally clipped snake segments through obstacles, likely due to inadequate collision margins, geometric approximations, and implementation issues. The original model's persistent geometry clipping, initialization complexity, and controller interference necessitated untangling several interconnected code files, which motivated the creation of a simplified three-link testbed. This minimal setup isolates essential simulation components, reduces initialization complexity by manually defining initial joint angles, and eliminates legacy controller artifacts. This stable environment is used and referred to in Chapter 4 for verifying the derived HPFC approach.

Technical Issues in the Simplified Model: One unresolved issue in the simplified simulation is a minor visual artifact in the form of a blue box that appears at specific ratios of snake link radius to link length. This box does not contribute to collision physics and is assumed to indicate the snake's origin. Preliminary debugging suggests that there are possibly unintended `geom` definitions, which require further manual inspection of the `xml` file. Debugging revealed several specific structural issues within the simplified model. They are summarized briefly as follows.

- *MuJoCo `xml` Format Schema Violations:* Incorrect attributes (`sensornoise`) caused loading failures. Corrected attribute names and assignment methods resolved these errors.
- *Incorrect Floor Orientation:* The floor plane was misaligned vertically due to a missing `zaxis` specification. Adding `zaxis="0 0 1"` resolved this.
- *Obstacle `Geom` Offsets:* Default positions unintentionally shifted obstacles. Explicitly setting `geom` positions (`pos="0 0 0"`) corrected this issue.
- *Obstacle Placement Errors:* An incorrect perpendicular angle calculation positioned obstacles incorrectly; adjusting angles from $+90°$ to $-90°$ corrected placement.
- *Contact Counting Errors:* Inflated contact counts resulted from counting all MuJoCo-generated contacts. Implementing object-pair filtering yielded accurate counts.
- *MuJoCo API Corrections:* Replaced deprecated quaternion normalization function (`mju_normalizeQuat`) with the correct API call (`mj.mj_normalizeQuat`).

The debugging process confirmed that the simplified model is suitable for stable, isolated controller verification.

Current Limitations and Open Problems

Despite the extensive simplifications, refactoring, and debugging of both simulations, some challenges remain.

- *Persistent XY Offset:* For longer configurations, unexplained lateral offsets remain at initialization, potentially due to overload of MuJoCo's physics solver, or visual-physical frame misalignment.
- *Coordinate Frame Mismatch:* Subtle but recurring mismatches between visual markers, `geoms`, and trajectory data introduce inconsistencies at initialization and during analysis.
- *Contact Detection Jitter:* When the snake body moves along an obstacle contacts appear to jitter. Contact losses and reestablishment oscillate at very high frequencies, as seen in Chapter 5. This is also transferred to the contact force measurement.

6.3.2 Development Issues of HPFC for the Simplified Model

Implementing HPFC for the 3-link simplified model was an iterative process that addressed several challenges related to model generation, controller logic, and actuator behavior. The most significant challenge was to accurately measure and interpret contact forces within the MuJoCo physics engine.

Phase 1: Contact Recognition: The initial phase focused on collecting contact information, such as location, magnitude, and direction.

MJCF Generation Error with `<flag>` Element:

Symptom: Python `TypeError` during MJCF generation: `Flag.__init__() got an unexpected keyword argument 'contactforce'`.

Cause: The version of the `dm_control` MJCF wrapper used does not accept `contactforce` as a constructor argument for `e.Flag`.

Solution: The script `generate_mjcf_main.py` was modified to set `contactforce` after object creation.
`flag = e.Flag(contact="enable"); flag.contactforce = "enable"`.

Status: Resolved.

Phase 2: Implementing Gait, Explicit Force Control, and Addressing Saturation: This phase involved adding explicit force control capabilities and addressing the resulting issues with the actuators.

Necessity to Control/Limit Contact Forces:

Symptom: Contact forces were excessively high, hindering motion. Initial HPFC lacked explicit force targets.

Solution: Introduced a PI force control loop for each contact normal, tracking `contact_force_reference_N`, with additional parameters (`use_explicit_force_control`, `Kp/Ki`). Motion is projected into the null space of force constraints.

Status: Implemented; revealed need for accurate force sensing and further tuning.

Actuator Force Saturation:

Symptom: Actuator outputs were persistently saturated at `maximum_torque_in_nm`; commanded torques exceeded limits.

Cause: Explicit force controller tried to correct large (possibly inaccurate) errors, resulting in excessive torques.

Diagnose: Temporarily increased torque limit to test controller behavior without saturation.

Strategy: Tune controller gains, reference values, and gaits; improve force accuracy.

Status: Identified; root causes (e.g., sensing inaccuracies) became progressively clearer.

Excessive Initial Contact Force Readings:

Symptom: Initial readings showed unrealistically high contact forces (e.g., 56N vs. 0.5N reference), suppressing projected motion.

Initially: (1) "Wedging" due to tight obstacle margins, no changes by increasing `margin`; (2) Saturation.

Refined: HPFC mistakenly used torque values instead of linear forces from `mj_contactForce`.

Solution: Corrected index (`[3:6]`) usage to extract linear force vector from `mj_contactForce`.

Status: Fixed internal force logic, but exposed that MuJoCo often reports near zero linear forces.

Unexpected Effect of `position_actuator_KP`:

Symptom: Modifying `position_actuator_KP` affected simulation despite using torque-based HPFC.

Cause: Both torque and position actuators were defined; `kp` gains of position servos were not disabled and interfered with torque commands.

Solution: Modified `init_controller` in `ctrl_snake.py` to set `kp` of position actuators to zero outside position mode.

Status: Resolved; torque motors now act as sole controllers when HPFC is selected.

Phase 3: Debugging Gait Execution and Deep Dive into Force Measurement: The focus of this phase was ensuring correct gait execution and resolving persistent contact force sensing issues.

"Rocking Motion Gait" Causing Wobbling:

Symptom:	Identical target angles at both joints caused C-shaped bending.
Cause:	`spatial_phase_per_joint_rad` was not introduced; no phase difference between joints.
Solution:	Set `spatial_phase_per_joint_rad` to π in `config.json`, inducing a 180°-phase shift.
Status:	Resolved; snake now performs alternating rocking motion as intended.

Inaccurate or Zero Contact Force Measurement (Multi-Stage Investigation):

Initial Symptom:	Logged contact forces were zero, even with clear penetration and visible MuJoCo force vectors.
Initial Hypothesis:	Transient contacts or late logging missed real force values.
Refined Cause:	`mj_contactForce` returned zero linear force components, while torque values were non-zero.
Intermediate Strategy:	Switched to reading `data.efc_force[contact.efc_address]` as a scalar force alternative.
Further Observation:	This value was still zero in many relevant cases (e.g., for body-obstacle contacts).
Root Cause:	Dominant EFC force components were sometimes resolved in other indices (e.g., `+1`, `+2`), due to contact orientation.
Final Solution:	Reconstruct total 3D world-frame constraint force from all Equality Force Constraint (EFC) components and project it onto the geometric contact normal $\mathbf{n}_i$ to get a usable scalar for HPFC and logging.
Status:	Resolved for static cases; internal and external force readings now reliable for all contacts.

Phase 4: Addressing no Contacts at Initialization: The underlying issue of this phase is that the HPFC does not establish contact in pure force control mode.

Symptom: With motion gains disabled (`Kp_motion = 0`, `Kv_motion = 0`) and force control active, the snake did not move to establish contact if it did not start in contact. Force plots remained zero.

Cause: Without PD motion input and starting from zero measured force, the controller lacks directional guidance to actively seek contact. Resulting torques may be ineffective or misdirected.

Strategy: A two-phase HPFC control concept was introduced:
Phase A (Positioning): Moderate motion gains (`Kp_motion_phaseA`, `Kv_motion_phaseA`) and a static and symmetric pose (`target_angles_rad`, e.g., `[0.0, 0.0]`) designed to establish contact. Force control temporarily disabled or set to zero.
Phase B (Force Control): Switch to low/zero motion gains. Enable explicit PI force feedback with desired references and tuned gains.

Status: Implementation logic added in `ctrl_snake.py`.

Challenges and Future Directions for HOAL Development

Despite the progress made, there are still several challenges and areas for future work concerning the HOAL. A primary long-term objective is to enable smooth and effective forward locomotion under HPFC. While it has been demonstrated that oscillations between states are possible, traversing different objects remains to be shown.

Another area of concern is the sensitivity of the system to over-constraint. Current multi obstacle configurations may activate multiple strong constraints simultaneously, creating an extremely narrow or degenerate motion null space. This can cause the snake to become stuck or unable to make significant adjustments. The rising k-value test of Section 5.2.2 demonstrated this issue, as illustrated in Figure 5.7. Addressing this problem will require strategic adjustments to obstacle placement or the development of constraint-management mechanisms within the controller. A related challenge involves the resilience of the force integral term `Ki_force_contact`. During initial testing, this gain was kept close to zero to prevent simulation instability. Future work may attempt to utilize this term more actively to eliminate steady state errors. However, this ap-

proach requires careful anti windup design and attention to behavior during transient contacts. Aggressive integration could lead to oscillations or overshooting in this situation. If despite improved force reconstruction, the HPFC still struggles to achieve stable regulation, a more in depth review of the MuJoCo contact model parameters (`solimp`, `solref`) may be necessary. These parameters influence the softness and damping of contact interactions. Tuning these parameters, particularly for snake and obstacle geometries, may help mitigate cases where excessively stiff solver behavior interferes with torque based control. Currently, the strategy favors working with MuJoCo's default or simplified solver settings and focuses on interpreting the available constraint force signals as accurately as possible.

Finally, general stability and robustness will remain central concerns in the tuning process. This includes suppressing residual jitter or oscillations, ensuring that torque commands stay within physical limits, and refining control gains (`Kp_motion`, `Kv_motion`, `Kp_force_contact`). The jitter in the simulation might be the consequence of a programming issue in the simulator or exaggerated realistic vibrational behavior. It may not be an issue of tuning the controller itself. Many settings were tested, but none could eliminate this phenomenon without severely violating the assumptions. Additional components, such as force signal filtering or the adjustment of the regularization factor `lambda_regularization_Jn_force`, may play an important role in ensuring long-term performance and reliability of the HOAL framework. Future adjustments for realistic simulation stability, including friction, joint damping, and armature parameters, need to be tested as well.

6.3.3 Current State of *SimSerpent*

The simplified 3-link setup currently provides a stable foundation, correctly initialized poses, and reliable basic plotting of positions, states, kinematics, forces, and contacts for static poses. However, in dynamic situations, an unresolved issue involving spiking contact forces and chattering obstacle-snake contacts persists, as discussed in the previous section and in Section 6.1. Simple control commands can be performed in both simulators, and HPFC was proven to work in the simplified setup, even with unresolved jitter when sliding along obstacles.

Section 4.1 addresses the general functionality of the MuJoCo simulation, which is depicted in Figure 4.1. Figure 6.6 provides an extended structural diagram, illustrating the code modules currently in use, to clearly distinguish between the original and simplified simulation structures. The green boxes in this figure correspond only to the version in which the complex trajectory is integrated. In the simplified 3-link model, these blocks are combined into a block called `plot_data.py`. A more detailed description of the presented blocks and the overall structure of the project can be found in the Appendix. The accompanying `README` files offer comprehensive insights into the different versions.

In summary, the substantial refinement and debugging of the *SimSerpent* simulator produced a more reliable, simplified setup, which is suitable for isolated HPFC and HOAL validation. However, additional work is necessary to integrate more complex test scenarios. Significant progress has been made toward achieving resilient and reproducible controller testing. Nevertheless, further refinements are essential to support extended or real-world applications.

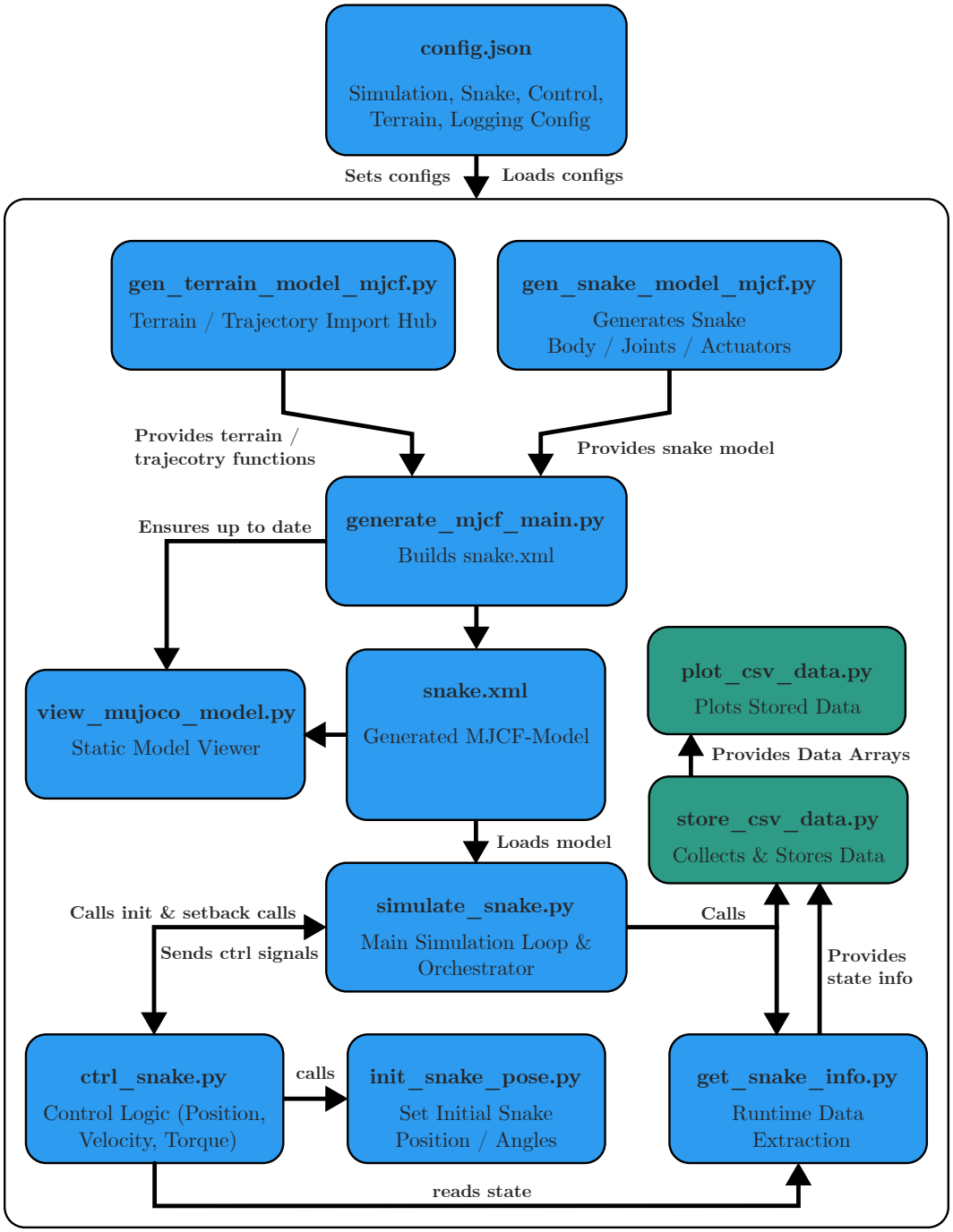


Figure 6.6: Structural diagram of the adjusted *SimSerpent* code.

7 Conclusion

This thesis investigated the application of Hybrid Position and Force Control (HPFC) to a simulated snake robot within the framework of Obstacle-Aided Locomotion (OAL). Building on the Hybrid Obstacle-Aided Locomotion (HOAL) concept, the work focused on developing and evaluating a passivity-based kinematic HPFC strategy, implemented in a modular and expandable simulation environment. The central research question, whether HPFC can be used to reliably navigate a snake robot through an obstacle-rich environment, was approached by integrating theoretical control strategies with practical simulation tools and validating their performance through carefully selected experiments.

To this end, a comprehensive modeling pipeline was developed, including the derivation of dynamic equations, construction of task and contact Jacobians, and a dedicated control architecture within the *SimSerpent* framework. Initial simulations focused on static form-closure scenarios to validate the ability of the controller to maintain commanded contact forces while preserving the robot pose. These tests demonstrate that the HPFC controller can successfully decouple motion and force objectives under static conditions, even in the presence of abrupt changes in force commands. Subsequent dynamic experiments were conducted to evaluate the controller performance under periodic motion with modulated force inputs. Despite observable unknown effects in the contact force measurements, most likely due to limitations in the simulation backend, the results indicate that the controller consistently maintained stable oscillatory movement patterns and show predictable behavior under both stepped and continuously rising force activations. The experiments further reveal key characteristics of the HPFC strategy, such as its resilience to transient disturbances and its ability to prioritize force regulation over motion as force demands increase. These findings confirm that the controller behaves as expected and aligns with theoretical predictions, thereby validating the underlying approach.

Beyond experimental validation, the thesis also contributes substantial technical development to the simulation infrastructure. Through multiple iterations, the *SimSerpent* simulator was refined to support simplified test cases, parameterized trajectory generation, and improved contact evaluation. The debugging process and implementation challenges were thoroughly documented, revealing key insights into the intricacies of integrating an advanced control logic into a dynamic, contact-rich simulation environment. Notably, the introduction of a two-phase control strategy proved instrumental in enabling reliable transitions between pose initialization and force regulation.

While the primary evaluation was limited to a simplified three-link configuration, the results present a foundation for future research, involving more complex trajectories and extended locomotion behaviors. Several limitations were identified, particularly in the areas of contact modeling, force sensing, and simulator fidelity. However, none of these compromised the core functionality of the control strategy. Instead, they highlight areas in which future work can build upon the current results to further improve accuracy, realism, and control depth.

In conclusion, the work presented in this thesis demonstrates that HPFC can be successfully implemented and evaluated within a physics-based simulation framework for obstacle-aided snake robot locomotion. Through a series of designed experiments, it was shown that the approach supports stable interaction with known obstacles and that the robot can follow predefined trajectory commands under simplified conditions. While the results are encouraging, they represent only an initial step, and further work is needed to assess the robustness and the generalizability of the proposed method in more complex scenarios. The modular structure of the developed controller and simulation environment provides a framework for future extensions, including adaptive strategies and real-world applications. With continued refinement, the proposed result has the potential to contribute meaningfully to resilient HPFC in navigating both simulated and physical snake robots through challenging environments.

8 Bibliography

[Ada14] Adafruit. Adafruit BNO055 Absolute Orientation Sensor Datasheet, 2014. Datasheet released in November 2014. Accessed: 2025-02-17. Retrieved from `https://cdn-shop.adafruit.com/datasheets/BST_BNO055_DS000_12.pdf`.

[Arn14] Carrie Arnold. Snake Robots Crack Mystery of How Reptiles Climb Dunes, 2014. National Geographic, Accessed: 2025-06-16.

[Bue19] Jake Buehler. How Sea Snakes, Surrounded by Salt Water, Quench their Thirst, 2019. National Geographic, Accessed: 2025-06-16.

[Dyn] Dynamixel XH540-V150R Servo Datasheet. Accessed: 2025-02-17. Retrieved from `https://www.robotis.us/dynamixel-xh540-v150-r/?srsltid=AfmBOoonhPjH6Yi72Uz-t5ydppTBhu836hfv7ScLoS8tUy5kitzeT8sd`.

[FSK98] Roy Featherstone, Stef Sonck, and Oussama Khatib. A General Contact Model for Dynamically-Decoupled Force/Motion Control. In *Experimental Robotics: International Symposium Barcelona*, pages 128–139. Springer, 1998.

[GLPS25] Irja Gravdahl, Jostein Løwer, Kristin Ytterstad Pettersen, and Øyvind Stavdahl. Form Closure-Based Path-Planning in Obstacle-Aided Locomotion for Snake Robots. Manuscript under review, 2025.

[GSK+22] Irja Gravdahl, Øyvind Stavdahl, Atussa Koushan, Jostein Løwer, and Kristin Ytterstad Pettersen. Modeling for Hybrid Obstacle-Aided Locomotion (HOAL) of Snake Robots. In *Vienna International Conference on Mathematical Modelling MATHMOD*, volume 55, pages 247–252. Elsevier, 2022.

[GST12] Rafal Goebel, Ricardo Sanfelice, and Andrew Teel. *Hybrid Dynamical Systems: Modeling, Stability, and Robustness*. Princeton University Press, 2012.

[Hir93] Shigeo Hirose. *Biologically Inspired Robots: Serpentile Locomotors and Manipulators*. Oxford University Press, Inc., 1993.

[Kha01] Hassan Khalil. *Nonlinear Systems*. Prentice Hall, Upper Saddle River, NJ, USA, 3 edition, 2001.

[Kou20] Atussa Koushan. Hybrid Obstacle Aided Locomotion (HOAL) in Snake Robots. Master's thesis, NTNU, 2020.

[LGVS22] Jostein Løwer, Irja Gravdahl, Damiano Varagnolo, and Øyvind Stavdahl. Proprioceptive Contact Force and Contact Point Estimation in a Stationary Snake Robot. In *IFAC Symposium on Robot Control SYROCO*, volume 55, pages 160–165. Elsevier, 2022.

[LGVS23] Jostein Løwer, Irja Gravdahl, Damiano Varagnolo, and Øyvind Stavdahl. A Novel Model for Link Dynamics in Planar Snake Robots Using Internal Constraint Force Sensing. In *IEEE Conference on Control Technology and Applications*, pages 872–877, 2023.

[Lil11] Pål Liljebäck. *Modelling, Development, and Control of Snake Robots*. PhD thesis, NTNU, 2011.

[Løw23] Jostein Løwer. *Snakes on a Plane: Modeling, Estimation and Locomotion for Planar Snake Robots in Cluttered Environments*. PhD thesis, NTNU, 2023.

[LPS09] Pål Liljebäck, Kristin Ytterstad Pettersen, and Øyvind Stavdahl. Modelling and Control of Obstacle-Aided Snake Robot Locomotion Based on Jam Resolution. In *IEEE International Conference on Robotics and Automation*, pages 3807–3814, 2009.

[LPSG10] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. Hybrid Modelling and Control of Obstacle-Aided Snake Robot Locomotion. *IEEE Transactions on Robotics*, 26(5):781–799, 2010.

[LPSG12a] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. Snake Robots: Modelling, Mechatronics, and Control. *Advances in Industrial Control*, 2012.

[LPSG12b] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. Snake Robot Locomotion in Environments with Obstacles. *IEEE/ASME Transactions on Mechatronics*, 17(6):1158–1169, 2012.

[LPSG13a] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. *Development of a Mechanical Snake Robot for Motion Across Planar Surfaces*, pages 55–61. Springer, 2013.

[LPSG13b] Pål Liljebäck, Kristin Ytterstad Pettersen, Øyvind Stavdahl, and Jan Tommy Gravdahl. Lateral Undulation of Snake Robots: A Simplified Model and Fundamental Properties. *Robotica*, 31(7):1005–1036, 2013.

[LSPG14] Pål Liljebäck, Øyvind Stavdahl, Kristin Ytterstad Pettersen, and Jan Tommy Gravdahl. Mamba - A Waterproof Snake Robot with Tactile Sensing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 294–301, 2014.

[ME-25] ME-Meßsysteme. K3D40±50N Force Sensor, 2025. Datasheet released on 07 January 2025. Accessed: 2025-02-17. Retrieved from `https://www.me-systeme.de/en/k3d40-50n`.

[Mør23] Oscar Brunell Mørk. SimSerpent: A Physics-based Simulator for a Next Generation Snake Robot. Master's thesis, NTNU, 2023.

[NEI16] Shunsuke Nansai, Mohan Rajesh Elara, and Masami Iwase. Dynamic Hybrid Position Force Control using Virtual Internal Model to Realize a Cutting Task by a Snake-Like Robot. In *IEEE International Conference on Biomedical Robotics and Biomechatronics*, pages 151–156, 2016.

[PD25] Bhavik Patel and Santosha Dwivedy. 3D Dynamics and Control of a Snake Robot in Uncertain Underwater Environment. *Robotica*, 43(1):1–28, 2025.

[Pet17] Kristin Ytterstad Pettersen. Snake Robots. *Annual Reviews in Control*, 44:19–44, 2017.

[Raf25] John Rafferty. Flying Snake, 2025. Encyclopedia Britannica, Accessed: 2025-06-16.

[RB98] Elon Rimon and Joel Burdick. Mobility of Bodies in Contact. A 2nd-order Mobility Index for Multiple-Finger Grasps. *IEEE Transactions on Robotics and Automation*, 14(5):696–708, 1998.

[RC81] Marc Raibert and John Craig. Hybrid Position/Force Control of Manipulators. *Journal of Dynamic Systems, Measurement, and Control*, 103(2):126–133, 1981.

[RLT+21] Jennifer Rieser, Tai-De Li, Jessica Tingle, Daniel Goldman, and Joseph Mendelson. Functional Consequences of Convergently Evolved Microscopic Skin Features on Snake Locomotion. *Proceedings of the National Academy of Sciences*, 118(6):e2018264118, 2021.

[Sc25] Øyvind Stavdahl and collaborators. Boa Snake Robot Main Design Document (Extract). Unpublished internal document. Preliminary version (2025-02-17), 2025.

[SHV06] Mark Spong, Seth Hutchinson, and Mathukumalli Vidyasagar. *Robot Modeling and Control*, volume 3. Wiley New York, 2006.

[SSL18] Filippo Sanfilippo, Øyvind Stavdahl, and Pål Liljebäck. SnakeSIM: A ROS-based Control and Simulation Framework for Perception-Driven Obstacle-Aided Locomotion of Snake Robots. *Artificial Life and Robotics*, 23:449–458, 2018.

[SSVO09] Bruno Siciliano, Lorenzo Sciavicco, Luigi Villani, and Giuseppe Oriolo. *Robotics: Modelling, Planning and Control*. Advanced Textbooks in Control and Signal Processing. Springer, 2009.

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A Physics Engine for Model-Based Control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[TFL13] Aksel Andreas Transeth, Sigurd Fjerdingen, and Pål Liljebäck. Snake Robot Obstacle-Aided Locomotion on Inclined and Vertical Planes: Modeling, Control Strategies and Simulation. In *IEEE International Conference on Mechatronics*, pages 321–328, 2013.

[TJL24] Bo-Ru Tseng, Jun-Yi Jiang, and Ching-Hung Lee. Adaptive Position/Force Controller Design Using Fuzzy Neural Network and Stiffness Estimation for Robot Manipulator. *International Journal of Fuzzy Systems*, pages 1–15, 2024.

[TLG+08] Aksel Andreas Transeth, Remco Leine, Christoph Glocker, Kristin Ytterstad Pettersen, and Pål Liljebäck. Snake Robot Obstacle-Aided Locomotion: Modeling, Simulations, and Experiments. *IEEE Transactions on Robotics*, 24(1):88–104, 2008.

[TP06] Aksel Andreas Transeth and Kristin Ytterstad Pettersen. Developments in Snake Robot Modeling and Locomotion. In *IEEE International Conference on Control, Automation, Robotics and Vision*, pages 1–8, 2006.

[WA85] Harry West and Haruhiko Asada. A Method for the Design of Hybrid Position/Force Controllers for Manipulators Constrained by Contact with the Environment. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 251–259, 1985.

[WAD+25] Yuexi Wang, Tianjiao An, Bo Dong, Mingchao Zhu, and Yuanchun Li. Adaptive Dynamic Programming-Based Finite-Time Optimal Backstepping Force/Position Control of Reconfigurable Robot Manipulators via Pareto Optimal. *IEEE Transactions on Automation Science and Engineering*, 22:10660–10671, 2025.

[Whi87] Daniel Whitney. Historical Perspective and State of the Art in Robot Force Control. *The International Journal of Robotics Research*, 6(1):3–14, 1987.

[Yos87] Tsuneo Yoshikawa. Dynamic Hybrid Position/Force Control of Robot Manipulators–Description of Hand Constraints and Calculation of Joint Driving Force. *IEEE Journal on Robotics and Automation*, 3(5):386–392, 1987.

[YS93] Tsuneo Yoshikawa and Akio Sudou. Dynamic Hybrid Position/Force Control of Robot Manipulators–On-Line Estimation of Unknown Constraint. *IEEE Transactions on Robotics and Automation*, 9(2):220–226, 1993.

[YST88] Tsuneo Yoshikawa, Toshiharu Sugie, and Masaki Tanaka. Dynamic Hybrid Position/Force Control of Robot Manipulators-Controller Design and Experiment. *IEEE Journal on Robotics and Automation*, 4(6):699–705, 1988.

Appendix

README.md 2025-05-21

Simplified Snake Simulator (3-Link Version with HPFC)

This repository demonstrates a simplified robotic snake simulator in MuJoCo, adapted from the SimSerpent project. This version focuses on a **3-link snake configuration** and implements a **Hybrid Position and Force Controller (HPFC)**.

Original repository credit: Master thesis 2023 by Oscar Brunell Mørk (Boa-Snake-Robot/SimSerpent).

How to clone repository (Original)

To clone the original SimSerpent repository (which this project is derived from), use:

```
git clone --recursive https://github.com/Boa-Snake-Robot/SimSerpent.git
```

Requirements

To run this code, you need the following dependencies:

- `mujoco`: The physics simulator.
- `numpy`: For numerical operations.
- `matplotlib`: For plotting simulation results.
- `pathlib`: For handling file paths (usually built-in).
- `glfw`: Required by `mujoco` for visualization.
- `dm_control`: Specifically for the `mjcf` library used to generate MuJoCo models programmatically.

Optional but recommended for environment management:

- `virtualenv`

You can install the required packages using pip:

```
pip install requirements
```

Virtual environment

It's recommended to set up a virtual environment before running the simulator to prevent version collisions between different Python packages. To enable a virtual environment, follow these steps:

1. Install virtualenv if you haven't already: `pip install virtualenv`.
2. Create a virtual environment (e.g., named `venv`): `virtualenv venv`.
3. Activate the virtual environment:
    - Windows: `venv\Scripts\activate`
    - macOS/Linux: `source venv/bin/activate`
4. Install the required packages within the activated environment (e.g., `pip install requirements`).
5. To exit the virtual environment, run the command: `deactivate`.

1 / 7

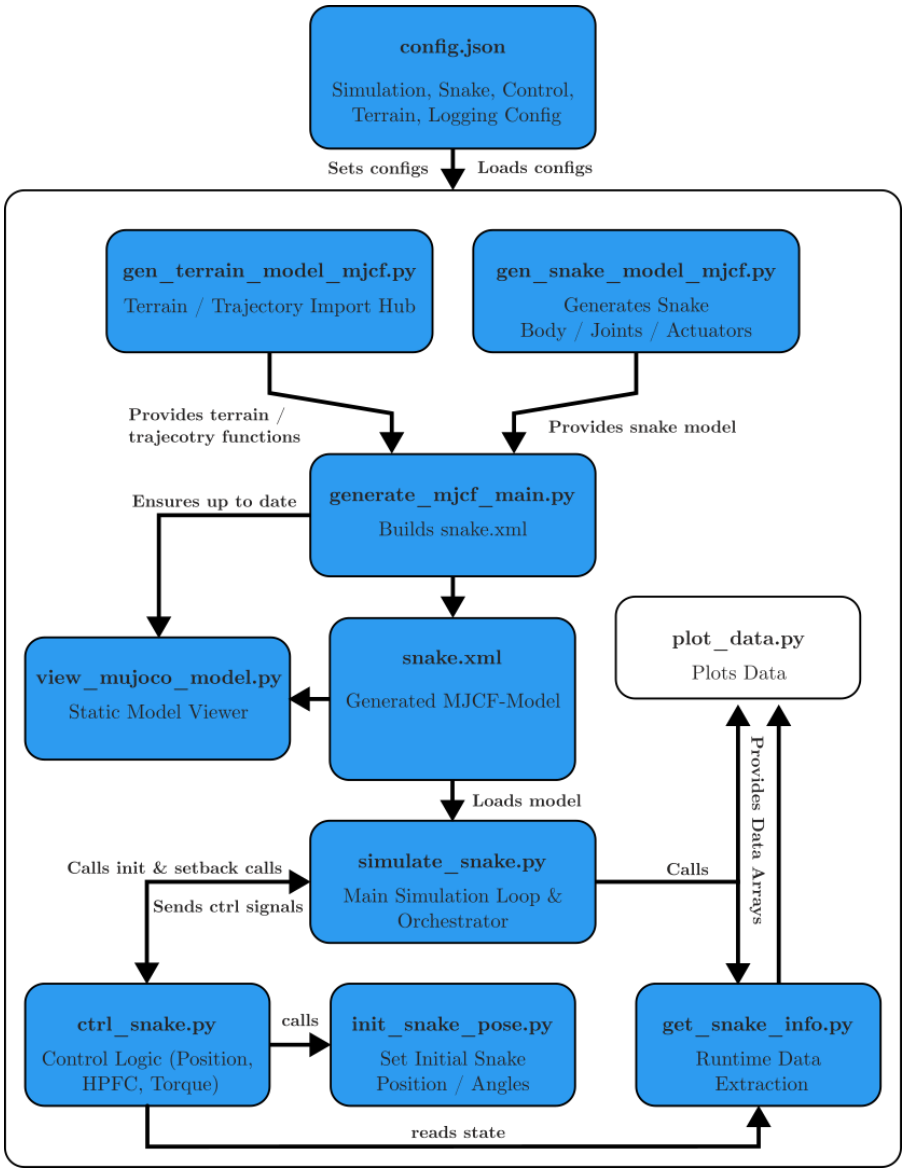
README.md 2025-05-21

Structure and Usage

The repository contains several Python scripts working together:

1. `simulate_snake.py`: The main script to run a simulation. It orchestrates model generation, simulation stepping, visualization, data collection, and plotting.
2. `generate_mjcf_main.py`: Automatically generates the MuJoCo model XML file (here `snake_simple.xml`) based on settings in `config.json` and helper modules. **This script is run automatically by `simulate_snake.py` and `view_mujoco_model.py`.** It specifically creates a **3-link snake model**.
3. `config.json`: Configuration file to control simulation parameters, snake properties, visualization options, terrain generation, and control settings (including HPFC parameters).
4. `view_mujoco_model.py`: A utility script to generate the model using `generate_mjcf_main.py` and then open it directly in the MuJoCo viewer. Useful for inspecting the static model without running a full simulation.
5. `snake_config.py`: Loads parameters from `config.json` and makes them accessible as Python variables for other scripts.
6. `ctrl_snake.py`: Handles the control logic for the snake. Implements:
    - **Hybrid Position and Force Control (`hybrid_force_position`)**: A two-phase controller. Phase A for initial positioning, Phase B for main gait execution with optional explicit force feedback and k-factor modulation.
    - **Position Control (`position`)**: Basic PD control using position servos.
    - **Torque Control (`test_torque`)**: Applies constant torques to joints.
7. `gen_snake_model_mjcf.py`: Helper script used by `generate_mjcf_main.py` to create the MJCF elements specifically for the **3-link snake body, joints, and actuators** (both position servos and torque motors).
8. `gen_terrain_model_mjcf.py`: Helper script used by `generate_mjcf_main.py` to generate terrain elements. Can generate specific cylinder obstacles. Uses `terrain_generation.py`.
9. `init_snake_pose.py`: Initializes the snake to a **predefined starting pose** specifically designed for the 3-link configuration (e.g., 45°, horizontal, -45° oriented segments).
10. `get_snake_info.py`: Contains functions to extract state information (kinematics, forces, angles) from the MuJoCo simulation data (`mjData`) during runtime. Used for data logging and plotting.
11. `plot_data.py`: Plots the collected simulation data using `matplotlib`. Includes plots for link kinematics, actuator torques, joint angles, snake-obstacle contacts, and total contact force magnitudes.
12. `geometry_utils.py`: Utility functions, currently including `get_quaternion_from_euler`.
13. `terrain_generation.py`: Contains helper functions for creating terrain elements, specifically `generate_cylinders` used by `gen_terrain_model_mjcf.py`. Includes option to visualize obstacle

centers.



Running a simulation

1. Modify `config.json` to set desired simulation parameters, snake properties, control method (e.g., `hybrid_force_position`), HPFC parameters, etc.
2. Run the main simulation script from your terminal:

```
python simulate_snake.py
```

This will first automatically run `generate_mjcf_main.py` to create/update the model XML file based on the current `config.json`, then launch the MuJoCo simulation window.

If the simulation is configured to plot data (`plot_data` in `config.json` is `true`), plots will appear after the simulation window is closed or the simulation finishes.

During the simulation, you can interact with the view using the mouse:

- **Left mouse button + drag**: Rotate the view.
- **Right mouse button + drag / Middle mouse button + drag**: Pan the view.
- **Scroll wheel**: Zoom in and out.

Keyboard interaction:

- **BACKSPACE**: Reset the simulation to the initial pose defined in `init_snake_pose.py` and re-initializes the controller.

Explanation of `config.json`

The `config.json` file controls various aspects of the simulation:

1. `xml_path` (string): Path/filename for the generated MuJoCo model file (e.g., "snake_simple.xml"). When changing the name the old file is kept, so different scenarios can be saved under different names.
2. `simulation_time` (float): Total duration of the simulation in seconds.
3. `simulation_timestep` (float): The physics time step (dt) for the simulation in seconds.
4. `integrator_type` (string): Physics integrator to use (e.g., "RK4", "Euler").
5. `plot_data` (boolean): If `true`, collects data during simulation and generates plots at the end.
6. `visual_options` (object): Controls visualization settings in the viewer.
    - `show_contact_force`, `show_contact_points`, `show_actuators`, `show_joints`, `show_center_of_mass`, `make_transparent` (booleans).
7. `snake_specifications` (object): Defines the snake's physical properties.
    - `nr_of_links` (integer): **Note:** While present, the generation scripts and controller in this version are hardcoded for **exactly 3 links**. This config value should be 3; other values might be ignored or cause warnings.
    - `link_mass_in_kg` (float): Mass *per link*.
    - `link_lenght_in_m` (float): Length of each link.
    - `link_radius_in_m` (float): Radius of the capsule representing each link.
    - `maximum_torque_in_nm` (array): Torque limits `[min_torque, max_torque]` for actuators.
    - `friction` (float): Friction coefficient for the snake's geoms.
    - `joint_damping` (float): Damping applied to the hinge joints.
    - `joint_armature` (float): Armature inertia added to the joints.
    - `sensor_noise` (boolean): Flag to enable/disable MuJoCo's sensor noise feature.
    - `snake_color_rgb` (array): Default RGBA color `[r, g, b, a]` for the snake links.
8. `control_parameters` (object): Configures the control system.
    - `control_method` (string): Selects the active control function from `ctrl_snake.py`. Supported:
        - `"hybrid_force_position"`: Activates the HPFC.

        - `"position"`: Activates basic position control.
        - `"test_torque"`: Activates constant torque application.
    - `test_torque_values` (array): `[torque_joint1, torque_joint2]` for `test_torque` mode.
    - `position_actuator_KP` (float): Proportional gain (Kp) for position servos in `position` mode.
    - **`hpfc_parameters`** (object): Parameters for the Hybrid Force/Position Controller.
        - `phase_A_duration_s` (float): Duration of the initial positioning phase (Phase A).
        - `Kp_gains_phaseA`, `Kv_gains_phaseA` (array): PD gains for Phase A.
        - `static_hold_gait_phaseA` (object): `target_angles_rad` (array) for Phase A.
        - `use_explicit_force_control_phaseA` (boolean): Enable explicit force control during Phase A.
        - `Kp_gains`, `Kv_gains` (array): PD gains for motion component in Phase B.
        - `gait_mode` (string): Gait pattern for Phase B (e.g., `"rocking_motion_gait"`, `"propulsive_wave_gait"`, `"static_hold"`).
        - `propulsive_wave_gait`, `rocking_motion_gait`, `static_hold_gait` (objects): Parameters for specific gait patterns (amplitude, frequency, target angles).
        - `k_activation_profile` (object): Modulates force reference over time in Phase B.
            - `k_values` (array): List of k-factors.
            - `k_segment_durations_s` (array): Durations for each k-factor segment.
        - `base_contact_force_ref_N` (float): Base target normal contact force (scaled by k-factor).
        - `Kp_force_contact`, `Ki_force_contact` (float): PI gains for explicit force control loop.
        - `force_integral_limit` (float): Limit for the force error integral.
        - `lambda_regularization_Jn_force` (float): Regularization for motion projection matrix.
        - `use_explicit_force_control` (boolean): Master switch for explicit force feedback component in Phase B.
9. **`terrain`** (object): Settings for generating obstacles.
    - `spawn_obstacles` (boolean): Master switch for obstacle generation.
    - `generate_specific_obstacles` (boolean): If `spawn_obstacles` is true, this controls generation of specific calculated cylinder obstacles.

Controlling the Snake Robot

Control logic is managed within `ctrl_snake.py`. The primary control method is the **Hybrid Position and Force Controller (HPFC)**, selected by setting `control_method: "hybrid_force_position"` in `config.json`.

Hybrid Position and Force Controller (`hybrid_force_position`)

This controller operates in two main phases:

- **Phase A (Initial Positioning):**
    - Duration defined by `phase_A_duration_s`.
    - Primarily uses PD control (`Kp_gains_phaseA`, `Kv_gains_phaseA`) to move joints to `static_hold_gait_phaseA.target_angles_rad`.
    - Can optionally include explicit force control (`use_explicit_force_control_phaseA`).
- **Phase B (Main Control/Gait Execution):**
    - Activates after Phase A.

- Combines a motion component (PD control based on a selected `gait_mode`) with an optional explicit force feedback component.
- **Motion Component:**
  - Gait pattern (e.g., "rocking_motion_gait", "propulsive_wave_gait", "static_hold") is selected by `gait_mode`.
  - Parameters for each gait (amplitude, frequency, target angles) are defined in their respective sections (e.g., `rocking_motion_gait`).
  - PD gains for this motion are `Kp_gains` and `Kv_gains`.
- **Force Feedback Component:**
  - Enabled by `use_explicit_force_control` (for Phase B).
  - Aims to achieve a `base_contact_force_ref_N` (dynamically scaled by `k_activation_profile`).
  - Uses a PI controller (`Kp_force_contact`, `Ki_force_contact`) based on measured contact forces.
  - Calculated force control torques are added to the motion control torques.
  - Motion torques are projected into the null space of contact Jacobians to minimize interference with force control objectives.
- **`k_activation_profile`**: Allows the `base_contact_force_ref_N` to be modulated over time during Phase B, enabling phased engagement or varying pressure.

Other Control Modes:

1. **Position Control (`control_method: "position"`)**:
   - Simple PD control using position servos. Target angles are currently fixed in `ctrl_snake.py` at `[-1.0, -0.5]` radians.
   - Kp gain from `position_actuator_KP`.
2. **Torque Control (`control_method: "test_torque"`)**:
   - Applies constant torques from `test_torque_values` to the torque motors.

Initialization:

- The snake's starting pose is set by `init_snake_pose.py`.
- The controller (including the HPFC instance) is initialized by `init_controller` in `ctrl_snake.py` when the simulation starts or is reset.

Explanation of how the snake model XML file is created

The MuJoCo model (`snake_simple.xml` by default) is automatically generated by `generate_mjcf_main.py` using the `dm_control.mjcf` library.

- `generate_mjcf_main.py` defines the overall MJCF structure (options, assets, defaults, worldbody, actuators, sensors).
- It calls helper functions from:
  - `gen_snake_model_mjcf.py` to create MJCF elements for the 3-link snake (bodies, geoms, joints, `torque_motor` and `position_servo` actuators).
  - `gen_terrain_model_mjcf.py` (which uses `terrain_generation.py`) to create obstacle geoms.

- Settings from `config.json` (timestep, link dimensions, friction, joint properties, actuator limits) are incorporated into the generated model.
- The `<default>` tag in `generate_mjcf_main.py` sets common properties for geoms and joints.
- The final XML is written to the file specified by `xml_path` in `config.json`.

Notes on Model and Controller Design

- The system is hardcoded for a **3-link snake** (tail, body1, body2) with 2 actuated revolute joints (yaw only).
- `get_snake_info.py` provides functions to access joint angles, velocities, body kinematics, and commanded torques.
- The HPFC logic in `ctrl_snake.py` includes calculation of contact Jacobians (`J_ci`), contact normals (`n_i`), and projection matrices (`P_motion`) for coordinating motion and force control.

How to add new controller types (Conceptual)

1. **Define New Control Function:**
   - In `ctrl_snake.py`, create a new Python function (e.g., `my_custom_controller(model, data)`).
   - Implement your control logic within this function. You can access simulation state via `data` and `get_snake_info.py` functions.
   - Set `data.ctrl[actuator_id]` for the desired actuators (e.g. `torque_motor`s for custom torque-based control).
2. **Register New Control Method:**
   - In `simulate_snake.py`, add an `elif` condition to the block that sets `controller_callback_func`:

```
elif control_method == "my_custom_method_name":
    controller_callback_func = cs.my_custom_controller
    print("Using My Custom Controller.")
```

3. **Update Configuration:**
   - In `config.json`, set `control_parameters.control_method` to `"my_custom_method_name"`.
   - Add any new parameters your controller needs under `control_parameters` or a sub-object. Access these in `ctrl_snake.py` by reading from the `config` dictionary.
4. **Update Initialization (if needed):**
   - If your controller requires specific initialization (e.g., setting different gains, initializing state variables), update `ctrl_snake.py::init_controller()`.

README2.md 2025-05-21

README: SimSerpent - Advanced Snake Robot Simulator

This repository contains the codebase for SimSerpent, an advanced robotic snake simulator built on MuJoCo. It allows for the generation of complex snake robot models of scaleable length, sophisticated trajectory definition, basic control strategies and detailed data logging and visualization.

This README provides an overview of the project structure, explains the role of key files, and details how to configure and run simulations.

What's New / Key Features (Compared to the Basic Simulaton Setup)

This simulator incorporates several advanced features:

1. **Parameterized Curve Trajectory Generation (`trajectory_generation.py`):**
    - Defines complex 2D paths using a sequence of straight so called "m-lines" and circular arcs.
    - Allows for precise, continuous path definition rather than discrete waypoints.
    - Includes functions to precompute geometry, calculate arc centers, and get points along the parameterized curve.
2. **Solver-Based Snake Initialization on Trajectory (`init_snake_pose.py`, `trajectory_generation.py`):**
    - When `spawn_on_trajectory` is enabled, the snake's initial pose is determined by placing joints sequentially along the generated parameterized curve.
    - Uses `scipy.optimize.minimize_scalar` to find points at exact `link_length` intervals along the curve, ensuring accurate initial configuration.
    - Joint angles are calculated based on the geometry of the snake along this curve.
3. **Advanced MJCF Generation (`generate_mjcf_main.py`, `gen_snake_model_mjcf.py`, `gen_terrain_model_mjcf.py`):**
    - Dynamically generates the `snake.xml` MuJoCo model file based on extensive parameters in `config.json`.
    - Includes generation of visual markers and physical obstacles along the parameterized trajectory.
    - `gen_terrain_model_mjcf.py` acts as a hub for importing various terrain and trajectory generation modules.
4. **Modular Control System (`ctrl_snake.py`):**
    - Supports multiple control methods selectable via `config.json` (torque, position, velocity, intVelocity).
    - Includes logic for specific experiments like "form closure" (often involving PID control) and "sinusoidal" motion.
5. **Comprehensive Configuration (`config.json`):**

- A central JSON file drives nearly all aspects: simulation parameters, snake physical properties, visual settings, control gains, terrain types, trajectory parameters, and data logging options.

6. **Detailed Data Logging and Plotting (`store_csv_data.py`, `plot_csv_data.py`):**
   - Extensive options for logging various simulation data (positions, velocities, forces, angles, etc.) to CSV files.
   - Automated plotting of key metrics at the end of the simulation.

File Structure and Descriptions

Core Simulation and Orchestration

`simulate_snake.py` - Main Simulation Runner

- **Purpose:** Entry point for running the full simulation.
- **Functionality:**
  1. Loads settings from `config.json`.
  2. Calls `generate_mjcf_main.py` (as a subprocess) to create/update `snake.xml`.
  3. Initializes MuJoCo model (`mj.MjModel`), data (`mj.MjData`), and visualization (GLFW window, camera, scene).
  4. Sets up GLFW callbacks for user interaction (camera control, simulation reset).
  5. Calls `ctrl_snake.init_controller()` to set actuator gains.
  6. Calls `init_snake_pose.init_snake_pos()` (via `ctrl_snake`) to set the snake's initial pose.
  7. Assigns the MuJoCo control callback (`mj.set_mjcb_control`) based on `config.json["control_parameters"]["control_method"]`.
  8. Runs the main simulation loop:
     - Steps physics (`mj.mj_step`).
     - Calls `store_csv_data.store_csv_data()` if logging is enabled.
     - Updates and renders the scene.
     - Handles GUI events.
  9. After the loop, calls `store_csv_data.write_to_csv()` if logging was enabled.
  10. Calls `plot_csv_data.plot_link_data()` if plotting is enabled.
- **Dependencies:** `config.json`, `generate_mjcf_main.py`, `ctrl_snake.py`, `store_csv_data.py`, `plot_csv_data.py`, `get_snake_info.py`, `init_snake_pose.py` (indirectly), MuJoCo libraries.

`config.json` - Master Configuration File

- **Purpose:** Centralized JSON file to control all aspects of the simulation, snake design, control, and terrain.
- **Key Sections:**
  - `xml_path`, `simulation_time`, `simulation_timestep`, `integrator_type`, `spawn_on_trajectory`, `plot_data`: General simulation settings.
  - `visual_options`: MuJoCo visualization flags.
  - `snake_specifications`: Physical properties of the snake (links, mass, dimensions, torque limits, color, default spawn if not on trajectory).
  - `control_parameters`: Active `control_method`, gains for various actuator types (`*_KP`), PID parameters.

  - `terrain`: Flags for enabling different terrain generation methods, including `generate_trajectory` (for the parameterized curve visualization/obstacles) and `spawn_obstacles`.
  - `prebuilt_experiments`: Toggles for specific experimental behaviors in `ctrl_snake.py` (e.g., `form_closure`, `do_sinus`).
  - `store_data_to_csv`: Granular control over which data variables are logged.

`view_mujoco_model.py` **- Simple Model Viewer**

- **Purpose:** Quickly visualize the generated `snake.xml` without running the full simulation dynamics.
- **Functionality:**
  1. Runs `generate_mjcf_main.py` (as a subprocess) to ensure `snake.xml` is current.
  2. Loads `snake.xml` into an `mj.MjModel`.
  3. Launches MuJoCo's interactive `viewer.launch(model)`.
- **Dependencies:** `config.json`, `generate_mjcf_main.py`, MuJoCo libraries.

MJCF Model Generation

`generate_mjcf_main.py` **– MJCF Model Generator**

- **Purpose:** Programmatically generates the `snake.xml` MuJoCo model file.
- **Functionality:**
  1. Reads parameters from `config.json`.
  2. Constructs the MJCF tree using the `mjcf` library (from `dm_control`).
  3. Sets top-level MJCF options (`<option>`, `<size>`, `<default>`).
  4. Includes assets like textures and materials.
  5. Calls `gen_snake_model_mjcf.generate_snake()` and related functions to build the snake robot structure (bodies, joints, geoms, actuators, sensors).
  6. Handles terrain and trajectory elements:
     - If `config.json["terrain"]["generate_trajectory"]` is true:
       - Uses functions from `gen_terrain_model_mjcf` (which imports from `trajectory_generation`) to:
         - `precompute_geometry`, `compute_arc_centers`.
         - Generate and add visualization markers (`generate_trajectory_visual_markers`).
         - Generate and add physical obstacles if `spawn_obstacles` is true (`generate_trajectory_obstacle_cylinders`).
     - Else (for simpler terrains): Calls functions like `generate_cylinders`, `generate_random_cylinders` from `gen_terrain_model_mjcf` (which imports from `terrain_generation`).
  7. Adds world elements like lighting and a floor.
  8. Writes the complete XML structure to `snake.xml`.
- **Dependencies:** `config.json`, `gen_snake_model_mjcf.py`, `gen_terrain_model_mjcf.py`, `mjcf` library.

`gen_snake_model_mjcf.py` **– Snake Robot MJCF Structure**

- **Purpose:** Defines the MJCF elements specifically for the snake robot.

- **Key Functions:**
  - `generate_snake()`: Creates and links all body segments.
  - `generate_linkage(nr_in_linkage)`: Creates a single link (`<body>`):
    - Head (link 0): Includes a `<freejoint>`, capsule geom, and sensor site.
    - Body links: Include a relative position, capsule geom, hinge `<joint>`, and sensor site. Geometry uses half-length (`link_length_m / 2.0`) and is offset appropriately.
  - `generate_actuators()`: Creates `<motor>` (torque), `<position>`, `<velocity>`, and `<intvelocity>` actuators for each hinge joint.
  - `generate_sensors()`: Creates `<velocimeter>`, `<accelerometer>`, `<gyro>` sensors for each link.
- **Dependencies:** `config.json` (for snake specs), `mjcf` library, `colors.py`.

`gen_terrain_model_mjcf.py` – Terrain & Trajectory Generation Hub

- **Purpose:** Acts as an import hub, centralizing access to various terrain and trajectory generation functions for `generate_mjcf_main.py`. It itself does not contain the core generation logic but imports it from other specialized modules.
- **Imported Functionality from:**
  - `trajectory_generation.py`: For parameterized curve definition, point generation, visualization markers, and trajectory-based obstacles.
  - `terrain_generation.py`: For basic cylindrical obstacles.
  - `geometry_utils.py`: For geometric calculations.
  - `snake_config.py`: For global configuration values.
  - `mjcf` library.
- **Provides these functions to `generate_mjcf_main.py` which then calls them.**

Trajectory and Geometry

`trajectory_generation.py` – Parameterized Curve Trajectory

- **Purpose:** Implements the logic for defining and using complex, continuous 2D trajectories made of straight "m-lines" and circular arcs.
- **Key Functions:**
  - `precompute_geometry(...)`: Calculates m-line start/end points, centers (`P_list`), and orientations (`phi_list`) based on circle parameters (`R_P`, `N_P`, `ALPHA_DEG`) and m-line half-length (`M_LINE`, typically `link_length_m`).
  - `compute_arc_centers(...)`: Calculates centers and radii of connecting circular arcs.
  - `parameterized_point(alpha, geometry)`: Returns a 2D point on the full curve for a normalized parameter `alpha` [0,1), handling interpolation on lines and arcs.
  - `compute_segment_lengths(...)`, `compute_cumulative_lengths(...)`: Calculate lengths of individual and cumulative segments.
  - `generate_snake_on_curve(...)`: Critically important for initialization. Generates `num_joints` points (typically `nr_of_links + 1`) along the curve, spaced by `link_length`, using `scipy.optimize.minimize_scalar` to find the precise `alpha` for each subsequent joint.
  - `generate_curve_visualization_points(...)`: Generates points for MJCF visual markers.
  - `generate_trajectory_visual_markers(...)`: Creates MJCF cylinder `<geom>` elements for visualizing the trajectory (non-colliding, visual group 3).

    - `generate_trajectory_obstacle_cylinders(...)`: Creates physical MJCF cylinder obstacles placed relative to the m-lines.
- **Dependencies:** `numpy`, `math`, `scipy.optimize.minimize_scalar`, `mjcf` library, `snake_config.py`, `geometry_utils.py`.

`geometry_utils.py` **– Geometric Utility Functions**

- **Purpose:** Provides core mathematical functions for 2D/3D geometry.
- **Key Functions:**
    - `get_quaternion_from_euler(...)`: Converts Euler angles to a MuJoCo quaternion.
    - `compute_distance(p1, p2)`: Calculates Euclidean distance.
    - `choose_perp(u, E, mode)`: Selects a 2D perpendicular vector.
    - `solve_for_r(E1, v1, E2, v2)`: Solves for intersection parameter `r`, used in arc center calculation.
- **Dependencies:** `numpy`.

`init_snake_pose.py` **– Snake Robot Initial Pose Setup**

- **Purpose:** Sets the snake's initial `data.qpos` (position, orientation, joint angles).
- **Functionality:**
    - If `config.json["spawn_on_trajectory"]` is true:
        1. Uses `trajectory_generation` functions (`precompute_geometry`, `compute_arc_centers`) to define the curve.
        2. Calls `trajectory_generation.generate_snake_on_curve()` to get `nr_of_links + 1` points along the curve, spaced by `link_length_m`.
        3. Sets head `qpos[0:3]` to the first point (plus small offset) and `qpos[3:7]` (orientation) based on the vector from the first to the second point.
        4. Calculates hinge joint angles `qpos[7:]` by finding the relative angle between consecutive segments formed by the generated points.
    - Else (default, form closure): Uses `set_default_snake_pose()` which sets a straight pose or a specific pre-defined pose for `form_closure` experiments, using parameters from `config.json`.
- **Dependencies:** `numpy`, `trajectory_generation.py`, `snake_config.py`, `geometry_utils.py`.

Controllers

`ctrl_snake.py` **– Snake Robot Controller Logic**

- **Purpose:** Implements various control strategies for the snake. Some of them are kept from the original repository and are not functional in this version.
- **Key Functions/Components:**
    - `init_controller(...)`: Sets actuator gains (`kp`, `kv`) for different servo types based on `config.json`. Initializes PID controllers if `form_closure` is active. (OLD)
    - `torque_controll(...)`: Implements direct torque control. If `form_closure` is true, uses PID controllers with time-varying setpoints.
    - `position_controll(...)`: Implements position servo control. If `do_sinus` is true, applies a sinusoidal wave pattern.
    - `velocity_controll(...)`: Placeholder for velocity servo control (currently empty).

- `intVelocity_controll(...)`: Implements IntVelocity servo control. If `form_closure` is true, uses target angles. (OLD)

Data Handling and Utilities

`terrain_generation.py` – Basic Cylindrical Terrain

- **Purpose:** Provides functions to create simple cylindrical obstacles.
- **Key Functions:**
  - `generate_cylinders(...)`: Creates cylinders at specified coordinates.
  - `generate_random_cylinders(...)`: Randomly places cylinders, avoiding snake spawn.
  - `generate_cylinder_path(...)`: Creates an alternating path of cylinders.
- **Dependencies:** `mjcf` library, `snake_config.py`, `random`.
- **Used by:** `gen_terrain_model_mjcf.py` for non-trajectory-based terrain.

`get_snake_info.py` – Runtime Snake Data Extraction

- **Purpose:** Retrieves specific snake state information from MuJoCo's `data` object.
- **Key Functions:**
  - `get_link_info(...)`: Position, velocity, acceleration, orientation, angular velocity of a link.
  - `get_joint_info(...)`: Joint angle and actuator forces.
  - `get_contact_info(...)`: Collision details for a link.
  - `get_force_info(...)`: External and internal forces/torques on a link.
  - `get_all_joint_angles(...)`: All hinge joint angles in degrees.
- **Dependencies:** `numpy`, `snake_config.py`.
- **Used by:** `simulate_snake.py` (for plotting/logging), `store_csv_data.py`, `ctrl_snake.py`.

`store_csv_data.py` – Simulation Data Recording to CSV

- **Purpose:** Collects simulation data during runtime and writes it to CSV files.
- **Key Functions:**
  - `store_csv_data(model, data)`: Called in sim loop. Appends current data (positions, velocities, forces, etc., based on `config.json` flags) to internal lists.
  - `write_to_csv()`: Called after sim loop. Writes data from internal lists to respective CSV files in a timestamped directory.
- **Dependencies:** `csv`, `get_snake_info.py`, `snake_config.py`.

`plot_csv_data.py` – Simulation Data Plotting

- **Purpose:** Uses Matplotlib to generate plots from simulation data.
- **Key Function:**
  - `plot_link_data(...)`: Takes arrays of time-series data (collected by `simulate_snake.py`) and creates a multi-panel plot (positions, velocities, forces, contacts, actuator efforts, etc.).
- **Dependencies:** `matplotlib.pyplot`, `snake_config.py`.

`snake_config.py` - Configuration Loader

- **Purpose:** Loads the `config.json` file into a Python dictionary (`config`) that is then imported and used by many other modules throughout the project.

- **Functionality:**
  1. Defines `_CONFIG_FILE_PATH`.
  2. Attempts to open and `json.load()` the file.
  3. Handles potential errors like `FileNotFoundError` or `json.JSONDecodeError`.
  4. Provides default values for many parameters using `config.get("key", "default_value")` if keys are missing from the JSON or if the file fails to load.
  5. Prints status messages about loading success or failure.
  6. Exposes key configuration values (e.g., `link_length_m`, `nr_of_links`) as module-level variables for easy import by other files.
- **Output:** A module-level `config` dictionary and several directly accessible config variables.
- **Dependencies:** `json`, `os`.

Running the Simulation

1. **Prerequisites:**
   - Python 3.x
   - Install required packages: `mujoco`, `numpy`, `matplotlib`, `pathlib`, `simple_pid`, `scipy`.

     ```
     pip install mujoco numpy matplotlib pathlib simple_pid scipy
     ```

   - Ensure `dm_control` (which provides the `mjcf` library) is installed if you need to modify MJCF generation beyond `config.json`. (Usually `pip install dm_control`)
2. **Configure:**
   - Edit `config.json` to set desired parameters for the snake, simulation, control, terrain, and data logging.
3. **Run:**
   - To run the full simulation:

     ```
     python simulate_snake.py
     ```

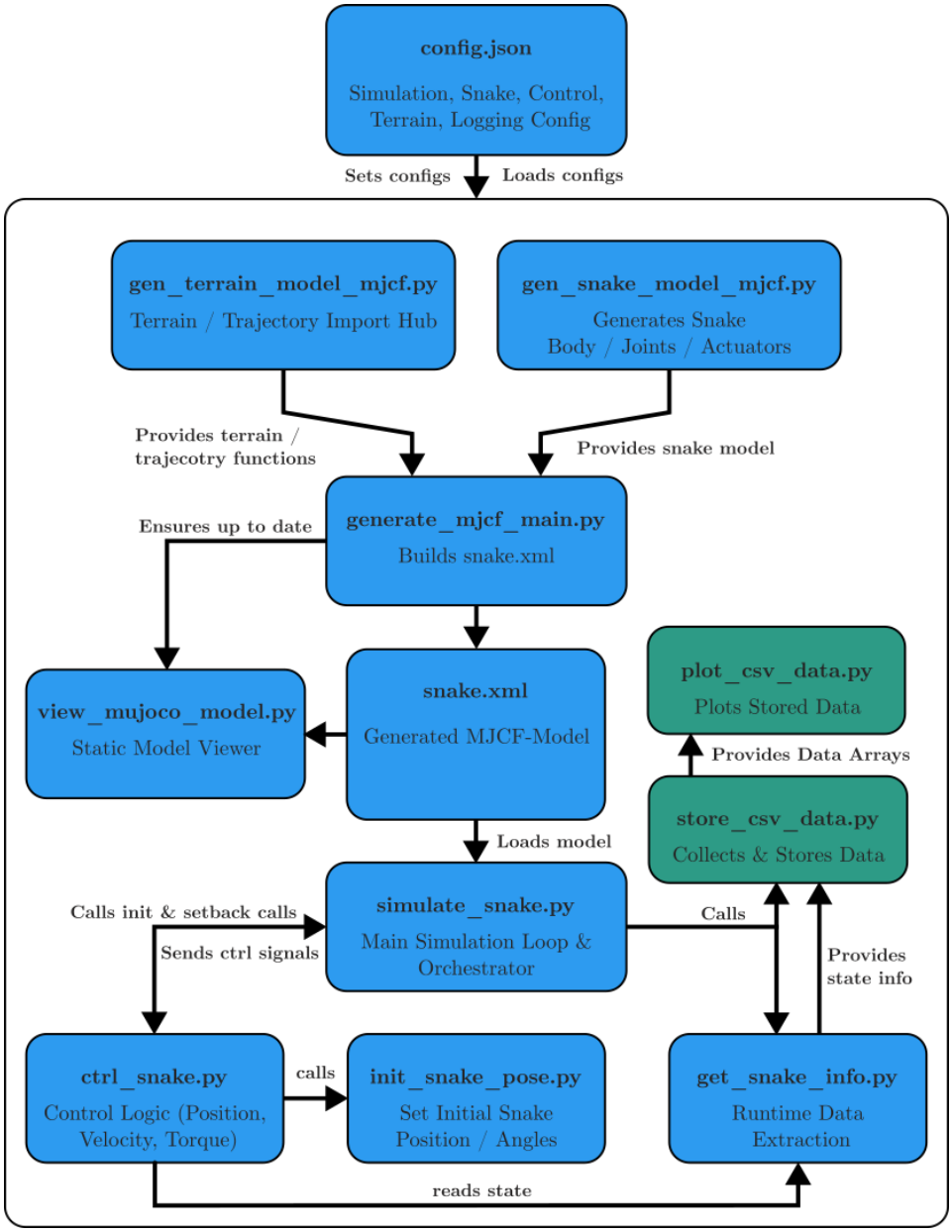
   - To just view the generated model:

     ```
     python view_mujoco_model.py
     ```

4. **Output:**
   - An interactive MuJoCo simulation window will appear.
   - If `store_data_to_csv["store_data"]` is true, CSV files will be saved in a new timestamped subdirectory within `csv_files/`.

- If `plot_data` is true, Matplotlib plots will be displayed after the simulation concludes.

UML (Conceptual Interaction Diagram)



Initializationstate of the Snake Robot

README2.md 2025-05-21