Hybrid Soft-Rigid Robots: Investigating Series and Parallel Configurations

by

Emily R. Sologuren

S.B., Electrical Engineering and Computer Science, Massachusetts Institute of Technology (2023)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by: Emily R. Sologuren
Department of Electrical Engineering and Computer Science
May 18, 2024

Certified by: Daniela Rus
Andrew (1956) and Erna Viterbi Professor, Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Hybrid Soft-Rigid Robots: Investigating Series and Parallel Configurations

by

Emily R. Sologuren

Submitted to the Department of Electrical Engineering and Computer Science
on May 18, 2024 in partial fulfillment of the requirements for the degree of

MASTER OF ENGINEERING IN ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

ABSTRACT

The diverse set of traits that soft-rigid robots possess have the potential to be applied towards a multitude of applications that require both strength and flexibility. This thesis looks at two kinds of soft-rigid robotic systems: the first is a series assembly of soft-rigid modules with stiffness modulation to form a soft-rigid robotic arm, and the second system is a parallel assembly of rigid bones casted into silicone to form a passive soft-rigid flipper for a robotic sea turtle.

We first introduce a new class of soft-rigid modules that can modulate their stiffness on a continuum through tendon-driven actuation and the integration of "soft" and "rigid" components. Their serial assembly form a self-standing, soft-rigid robotic arm (SRRA). When coupled with an adapted soft PD+ controller, we generate trajectories that demonstrate the manipulator's ability to deform for maneuvering tasks and stiffen for load-bearing tasks.

The robotic sea turtle's parallel, soft-rigid flippers emulate those of its animal counterpart. To leverage this structure for underwater locomotion, we look at a CPG-coupled reinforcement learning framework to optimize for a forward swimming gait.

Thesis supervisor: Daniela Rus
Title: Andrew (1956) and Erna Viterbi Professor

Acknowledgments

I would first like to thank my advisor, Daniela Rus, for her mentorship and unwavering support these past three years. I'm truly grateful that you have provided me the opportunity to create and explore new and exciting robotic systems in a welcoming lab environment. Your positive outlook, work ethic, and innovative thinking continue to inspire me.

There are two mentors I'd particularly like to thank. First, to Jim Bern, for mentoring me when I was a SuperUROP and shaping the beginnings of this research work. Your technical expertise coupled with your humor made the lab an educationally hilarious environment. I'm also grateful to my second mentor, Zach Patterson, whose investment in teaching and guiding me these past two years has led me to become a better researcher. Thank you both for your wisdom and guidance. I would also like to thank all the members of DRL, especially Steven, Lilly, Greg, Annan, Alex, Joseph, and John. You made the lab a fun and inviting place for me. And, of course, a special thank you to Mieke, without whom DRL would not operate.

I'd like to thank my entire family. A particular thanks to my parents, whose consistent love and support have gotten me to this point. To my dad, Federico Sologuren, who is still my engineering role model to this day. To my mom, Rosmery Sologuren, who always believed in me and encouraged me to pursue my passions. To my older siblings Melisa, Rumi, and Daniel: you guys were always a phone call away.

Finally, thank you to my friends both in and outside of MIT for all the adventures into Boston, road trips, and fun late nights.

Contents

List of Figures

Chapter 1

Introduction

Robots with soft-rigid structures have the potential to significantly broaden the capabilities and applications of robotic technology, especially within the fields of human-interaction and underwater locomotion. The role of robots in human-centric environments is still limited due to the requirement for them to be safe enough to handle unexpected collisions while being strong enough to carry out strenuous tasks. Similarly, effective underwater locomotion demands adaptable navigation and streamlined propulsion, ideally packaged within a lightweight system. Classical autonomous underwater vehicles (UAV) are typically bulky, require a considerable power source, and typically rely on noisy propulsion systems that make it difficult to closely study marine life without disturbing them. Soft robots are known for their intrinsic compliance and robustness, which provides them the flexibility to adapt in unstructured settings, as well as to safely interact with humans[1]. They are also considerably more light-weight than their rigid counterparts. However, these traits limit them in load-bearing situations. Rigid robots exhibit strength and precision; however, their typically heavy build often impedes safe interaction with humans, requiring a significant amount of autonomy. A promising approach to efficient locomotion is through the creation of bio-inspired robots capable of emulating the optimal locomotion observed in natural organisms. However, marine animals, such as sea turtles, have an anatomical structure that is not entirely soft or rigid.

We aim to leverage the potential of soft-rigid structures from a design and control perspective, investigating these structures in their serial and parallel formations, which each have their appropriate application. Serially linked soft-rigid robots hold potential in leveraging traits from its soft and rigid counterparts to carry our real-world tasks in a variety of environments. Indeed, there are several soft-rigid manipulators in the literature[2] that incorporate soft and rigid materials into their structure. One such example that this thesis expands on is from Bern et al.[3], which features a series of 2D soft-rigid modules that form a planar, soft-rigid manipulator capable of modulating between its "soft" and "rigid" states by increasing the stiffness of its modules via cable contraction.

Parallel soft-rigid structures, while less explored in robotics, hold a lot of promise for underwater locomotion, since its configuration emulates the natural parallel finger bone structure found in real-life sea turtle flippers [4]. We can potentially accomplish sea turtle-like swimming gaits by incorporating a similar rigid parallel structure into the silicone flippers of a robotic sea turtle.

While soft-rigid structures hold mechanical intelligence within the robot's body, it is only as effective as the control and learning algorithms driving their motion. Soft robots are continuum systems and thus infinite-dimensional, making them highly non-linear and underactuated. Consequently, unlike with rigid robots, there are still open questions on how to properly define the dynamics that model soft robots [5]. More interestingly, even with a proper model, there is still the control challenge of effectively leveraging a soft robot's softness at appropriate times. Notably, this includes their ability to store energy during unexpected collisions and their capacity to exhibit precise behavior.

For a soft-rigid manipulator, we need a proper dynamics model, coupled with a controller that leverages the shape and stiffness change of the robot's individual modules. For the robotic sea turtle, controlling its motion presents a challenge due to the intricate dynamics governing the interactions between the underwater environment and the robot. While simulations of underwater soft robots using hydrodynamic models exist [6], there still exists a gap between simulation and real-world performance, which limits the ability to reliably and autonomously navigate unstructured environments. Reinforcement Learning holds promise in addressing this problem, since it does not require prior knowledge and can improve its policy through interaction and sensor feedback. However, scaling a learning algorithm for a robotic sea turtle requires a proper abstraction of the problem space such that a policy can develop in an efficient and timely manner.

In this thesis, we focus on two soft-rigid robotic systems: one comprised of serially linked soft-rigid modules to construct a Soft-Rigid Hybrid Arm (SRHA), and the other involving the parallel arrangement of rigid links within a silicone body resembling a flipper for robotic sea turtle locomotion. For SRHA, we introduce the design, actuation, and assembly of soft-rigid modules, the integration of an adapted soft PD+ controller for shape control, and several experiments that demonstrate SRHA's ability to modulate its shape for maneuverability and strength. For the robotic sea turtle, we implement a Reinforcement Learning framework aimed at learning the parameters of Central Pattern Generators to optimize a forward swimming gait. This approach leverages the parallel soft-rigid structure of its locomotion without requiring direct prior knowledge of the flippers' dynamics.

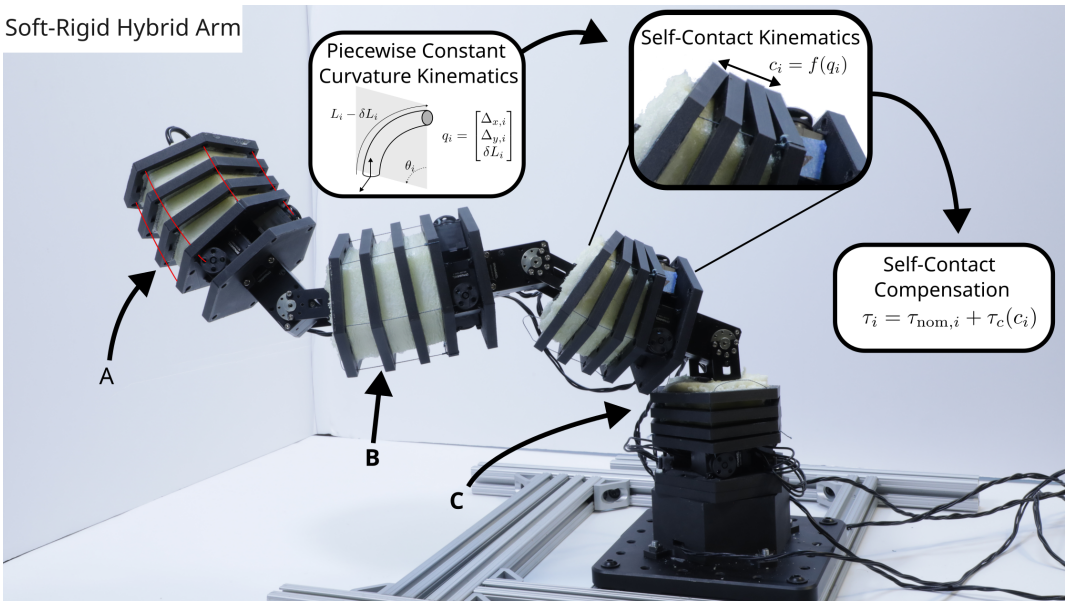


Figure 1.1: A modular soft-rigid hybrid arm that can operate as a series of rigid bodies or soft segments. The figure shows a control workflow that incorporates contact compensation based on forward kinematics calculations. (A) Each module is controlled by a set of three tendons (visible ones on the first module are highlighted in red). (B) When a module is uncompressed, the module acts like a soft segment, particularly under load. (C) When the module is completely compressed, it acts as a de facto rigid body.

1.1 Contributions

For the Soft-Rigid Hybrid Arm (SRHA), we contribute the following:

1. Design, fabrication, and characterization of the soft-rigid modules.
2. Assembly of soft-rigid modules into a self-standing manipulator.
3. Integration of electronics, software, and controller.
4. Design of hardware experiments featuring SRHA successfully maneuvering through an obstacle course for object retrieval.

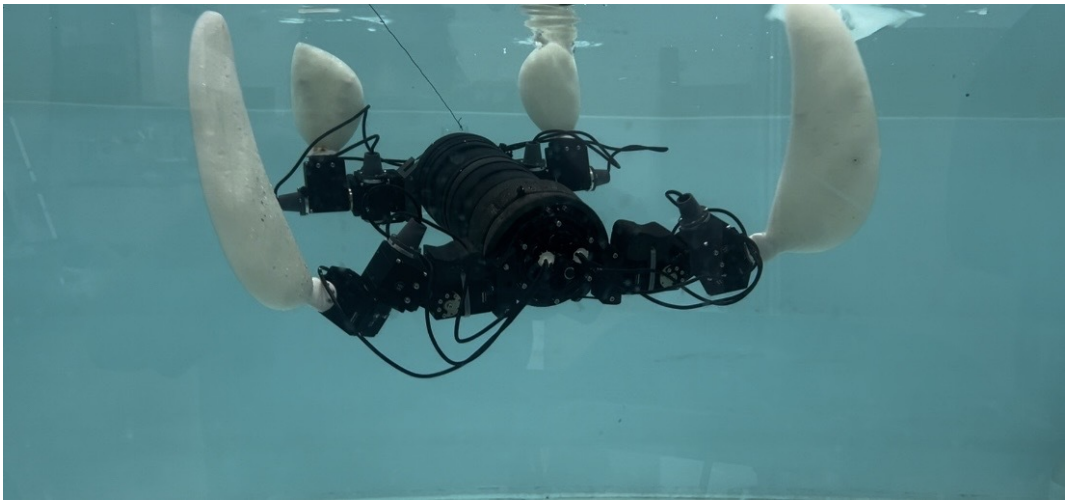


Figure 1.2: The robotic-sea turtle with soft-rigid flippers.

For the Robotic Sea Turtle with Soft-Rigid Flippers, we contribute the following:

1. Development of embedded software for motor control and learning.
2. Adaptation of policy-gradient algorithm that optimizes for CPG parameters for forward gait trajectories.
3. Simulation experiments for algorithm validation.
4. Preliminary experiments and demonstrations on robotic sea turtle platform.

1.2 Organization

Chapter 2 lays out the background of soft-rigid structures within robotics and how it has been applied so far. It also provides an overview of other existing soft robot solutions for arm manipulation, as well as motivation for bio-inspired underwater locomotion and approaches to learning locomotion.
Chapter 3 introduces the Soft-Rigid Hybrid Arm (SRHA) that applies soft-rigid structures in series. We outline the motivation, design, and fabrication. We then characterize the stiffness range of the soft-rigid modules, as well as provide a few hardware demonstrations of the robot applying its flexibility and load-bearing abilities.
Chapter 4 introduces an initial approach to integrating CPGs to a light-weight policy gradient method for learning a forward swimming gait on a robotic sea turtle platform with soft-rigid flippers.
Chapter 5 is a discussion of the work and lessons learned from both robotic platforms and future work.

Chapter 2

Related Work

2.1 Soft-Rigid Robots

There are several examples of soft-rigid robots in the literature. Notably, there is a single tendon, soft-rigid finger fabricated by casting a series of rigid bones into a soft silicone body [7]: this work supports the idea of internal rigid structures enabling soft robots to resist and apply larger forces as compared to the same silicone body without. Onal's Salamanderbot [8] is an example of using soft-rigid structures towards more flexible and adaptive locomotion by relying on a continuum origami-based design that is cable-driven such that it can actively bend to navigate maze-like environments. In addition to cable-driven actuation[7], [9], soft-rigid robots can achieve stiffness tuning from several other modes[10], including biologically-inspired[11], [12], pneumatic[13], [14], and vacuum [15] approaches. Pneumatically-driven, soft-rigid robots like in [13] rely on a series of pneumatic soft-robotic joints. When multiple are assembled, it creates a gripper that has high payload and dexterity while also being light-weight. Work in [11] relies on antagonistic actuation, whereby several actuators act on a joint such that the joint's state remains constant while the stiffness increases. However, these other methods of actuation are limited by a series of compressors or pumps, which makes the overall system bulky and limits their portability[16], which is especially important for untethered, biologically inspired robots that need to house their own electronics and power source within their body[17]. Recent work in the MIT Distributed Laboratory has looked into the design of soft-rigid modules that are capable of varying their stiffness and shape [3] by tendon-driven actuation, which applies force to a series of rigid plates to actively compress the soft-foam core. These modules were later linked in series to form a manipulator, allowing the system to perform a variety of motions in a planar setting.

New designs and actuation methods necessitate new control systems. While rigid-bodied robots can be modeled with a finite number of links and joints [18], soft-bodied robots possess a large number of degrees of freedom, requiring different kinematics, controllers, and planning algorithms [1]. The choice of model is particularly critical. Several closed-loop controllers [19]–[21] utilize models such as Piecewise Constant Curvature (PCC) [22] and discrete Cosserat models [23]. Additionally, there are approaches based on the finite element method (FEM) [24], [25]. For Bern's serially-linked 2D manipulator [3], we used an FEM-based soft robot simulator [26], adapted to handle contact when a module stiffens. However,

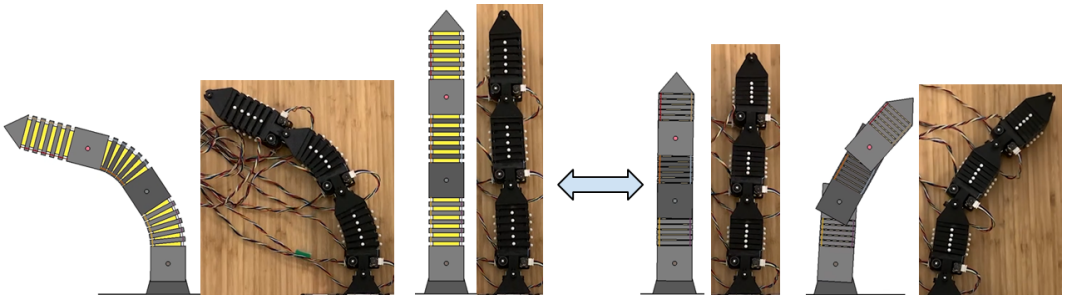


Figure 2.1: Overview of the 2D soft-rigid modules assembled in series to form a manipulator, where each module relies on two motors for compression and bending. This allows the manipulator to bend within its planar setting in two directions. This figure originally appeared in [3].

the high dimensionality of FEM-based approaches makes them challenging to implement in feedback controllers. Recent work on elastically coupled, parallel structures leverages inherent elasticity for indirect "sensing" of force and control [27]. Furthermore, there are reinforcement learning-based control methods [28] for cases where soft-robot modeling is too complex or computationally intensive.

2.2 Approaches to learning for underwater robotic systems

Studying marine life in a non-invasive way proves challenging for typical autonomous underwater vehicles (UAV); their mechanical structure and/or method of propulsion makes it difficult to blend into a natural underwater habitat without introducing some kind of disturbance. One avenue to address this problem has been the development of bio-inspired robots that match both appearance and locomotion of their underwater counterparts. A key to mimicking the behavior of biological systems has been the incorporation of soft materials into the structure [29], as was done at the MIT Distributed Robotics Laboratory with the soft-robotic fish [30]. The soft and compliant characteristics of this robotic system allowed the robot to replicate undulating fish tail motions commonly seen in real-life fish. This has inspired the exploration of systems that incorporate both soft and rigid materials to replicate motions for more complex sea animals, such as the sea turtle, which has more degrees of motion and therefore more complex swimming gaits that allow it to move efficiently underwater [31], [32] To accomplish proper underwater locomotion for a robotic sea turtle will require an algorithmic approach. Developing trajectory gaits for robotic systems with multiple degrees of freedom is still a challenging problem, especially for robots that work in underwater environments, where accurately capturing hydrodynamics requires computationally expensive or complex mathematical models [33]. Previous work has looked into using hydrofoils or central pattern generators (CPGs) to implement coupled nonlinear oscillators that model the undulation of a simple robot fish tail [34]–[36]. However, to our knowledge this has yet to be explored for more complex sea animals, such as the sea turtle.

Chapter 3

Robotic Arm with Soft-Rigid Modules: Series Configuration

3.1 Motivation

While there are several robotic arms that can operate in the real-world, most of them are restricted to controlled environments. They are limited in their ability to operate in more uncontrolled spaces, which require more flexibility, especially for scenarios that involve humans, such as hospitals and elderly homes. Soft robots' intrinsic compliance and softness guarantees both the ability to maneuver through complicated environments while posing no harm to humans. However, their softness limits their load-bearing capacity. A potential solution to this problem is relying on soft-rigid modules capable of varying their degree of rigidity. Implementing soft-rigid modules into a serially-linked manipulator can address the need for flexibility and safety that soft robots have while also meeting load-bearing requirements, since this would allow the manipulator to bend at certain parts of its body while also stiffening during points where it needs the strength to carry out a task.

In this chapter, we present a Soft-Rigid Hybrid Arm (SRHA) assembled from several soft-rigid modules capable of modulating their stiffness through the compression of rigid plates surrounding their soft, foam core. We also integrate an adapted version of a soft robot PD+ controller [19] to handle the self-contact experienced between the rigid plates. As seen in 3.4 and 3.5, this serial assembly of soft-rigid modules coupled with a reliable controller allows us to bend, compress, and rotate within the state space of SRHA to accomplish tasks that require both flexibility and strength.

The videos demonstrating the stiffness modulation along with the hardware experiments explained later in this chapter can be found under the 'videos' folder of this repository: https://github.com/ersolog/SRRA.

3.2 Methods

3.2.1 Design

The Soft-Rigid Hybrid Arm (SRHA) builds on work we presented in Bern et.al[3], which features a 2D manipulator concept of stiffness modulation inspired by a vintage push puppet toy. Push puppets rely on the tension of their cables to maintain an upright position. When their cables loosen, their structure collapses. This actuation idea is carried out in a similar vein for a soft-rigid module, where increasing tension increases the rigidity. This is done by inserting soft material between several rigid plates and routing cables controlled by servos. When the module is completely unactuated, the foam is uncompressed, allowing the soft-rigid module to bend or compress freely in any direction. As we contract the cables, the increasing tension in the cables pull the rigid plates closer together, compressing the soft foam and increasing the overall stiffness of the module. The module is fully compressed once its rigid plates make full contact, at which point the module can be treated as a rigid body. While this concept is featured within the 2D manipulator, the old design is limited in several ways. Firstly, the 2D module design is not omnidirectional: its two motor configuration constricts it to bend in two directions 2.1, thereby limiting the overall maneuverability of the fully assembled arm. Secondly, the mechanical structure of the manipulator is unable to support its own weight, limiting its operation to a planar setting. Lastly, the manipulator has no control implementation onboard, making it difficult to fully leverage its stiffness modulation to carry out real-world tasks.

The new design (SRHA) successfully removes these three constraints. As seen in B of 3.1, we expand the soft-rigid module design by adding an additional servo to the base of the soft-rigid module. This allows each module within the full assembly to independently bend freely in any direction. Each soft-rigid module has a hexagonal motor plate at the bottom, two standard hexagonal plates in the middle, and a joint motor plate at the top. All hexagonal plates have through-holes for cable routing. The bottom hexagonal motor plate is what mounts the three motors responsible for contracting the cables. The middle hexagonal plates have struts pointing inwards to allow the soft foam to adhere to during the foam casting process. The top joint motor plate has four screw holes at its center to allow us to mount a rigid joint motor in between modules. Therefore, when all modules are fully compressed, our full arm assembly mimics a rigid manipulator. For this thesis, we choose to incorporate four of these soft-rigid modules into a manipulator. We also integrate a motor at the robot's base to allow for full rotation about its vertical axis, resulting in a more redundant state space for SRHA to operate on.

3.2.2 Fabrication

All soft-rigid modules have their hexagonal rigid plates 3D printed from Markforged Onyx®, a micro carbon fiber filled nylon. Each hexagonal plate is 7mm thick, and each of its sides is 50mm in length. The hexagonal shape discretizes the shape of our modules, which simplifies the forward kinematics used for later work in [37]. The foam core is made of Smooth-On Flex Foam-iT! III Flexible Polyurethane Foam, and has a volume of 307 cm^3. We use a 3D printed mold which ensures a consistent 25mm distance between each rigid plate, thereby

ensuring uniform stiffness during compression. We also rely on the struts and inner parts of the hexagonal rigid plates for the foam to adhere to when casting. All other areas of the rigid plates are protected by spraying Ease Release 2831. Once all plates are inserted within the mold, we seal it and pour in 20mL of Smooth-On Flex Foam-iT! III solution and allow it to cure for 2 hours. After the curation period, we screw three XC330-T288-T Dynamixels onto the bottom motor plate of the module, where each motor has a 3D printed spool attached to its horn. Besides the module at the top of SRHA, we screw a FR12-H101K bracket onto the top joint motor plate of each module. The bracket connects to the horn of a XM430-W350-R Dynamixel motor, also known as the joint motor between modules. The bottom of each XM430-W350-R joint motor is screwed onto the bottom of all modules besides the bottom most module, which instead is mounted to the 3D printed rotating platform actuated with a XM430-W350-R mounted inside. Once fully assembled, the robot reaches a height of 0.68m and weighs 1.5kg.

We daisy-chain all XC330-T288-T module motors using 3-pin JST cables that connect to a Dynamixel Power hub, which connects to a small Dynamixel USB communication converter (U2D2). Similarly, all XM430-W350-R are daisy-chained to a second power hub and U2D2. All motors are powered on 12V, which in our case came from an off-the-shelf power supply. We note that we can replace our bench top power supply with a 12V LiPo battery to increase portability of the robot. The two U2D2s connect via micro-usb cable to a standard laptop. We use Dynamixel's SDK in Python to control the motors via serial packet communication.

The code used to control SRHA can be found in this public repository here. We have verified that the software works on Windows and Linux. To calculate the state of our manipulator in software, we keep track of all cable lengths by relying on the encoders embedded in the motors to measure changes in angle. Both the XC330s and XM430s have an embedded absolute encoder with a 12-bit resolution from AMS. During calibration, we contract the cables to the point where the cables slightly start initiating module contraction. We then measure the distance between each of the motor's spools to the top of the module and refer to this as the neutral or home cable lengths. Using the Dynamixel SDK API, we then call a reading of all the motor's angle measurements and store them as reference angles. Each reference angle can then be used to keep track of changes in angle as the module contracts. We rely on the change in angle measurement to calculate the cable lengths as $l_i = l_0 - (\Delta\theta_i r)$, where l_i is the current length of cable i, l_0 is the original cable length, $\delta\theta$ is the change in motor angle, and r is the radius of the spool. To ensure fast enough packet communication between the computer and motors, we us Dynamixel's Protocol 2.0 option, set the baud-rate to 2 Mbps, and manually set the USB latency timer on the computer's ports to 1 ms. This allows our control loop to run at 100-200 Hz. It is also required to use the Bulk Read and Bulk Write methods from Dynamixel's SDK to simultaneously read and send packets to both the module and joint motors.

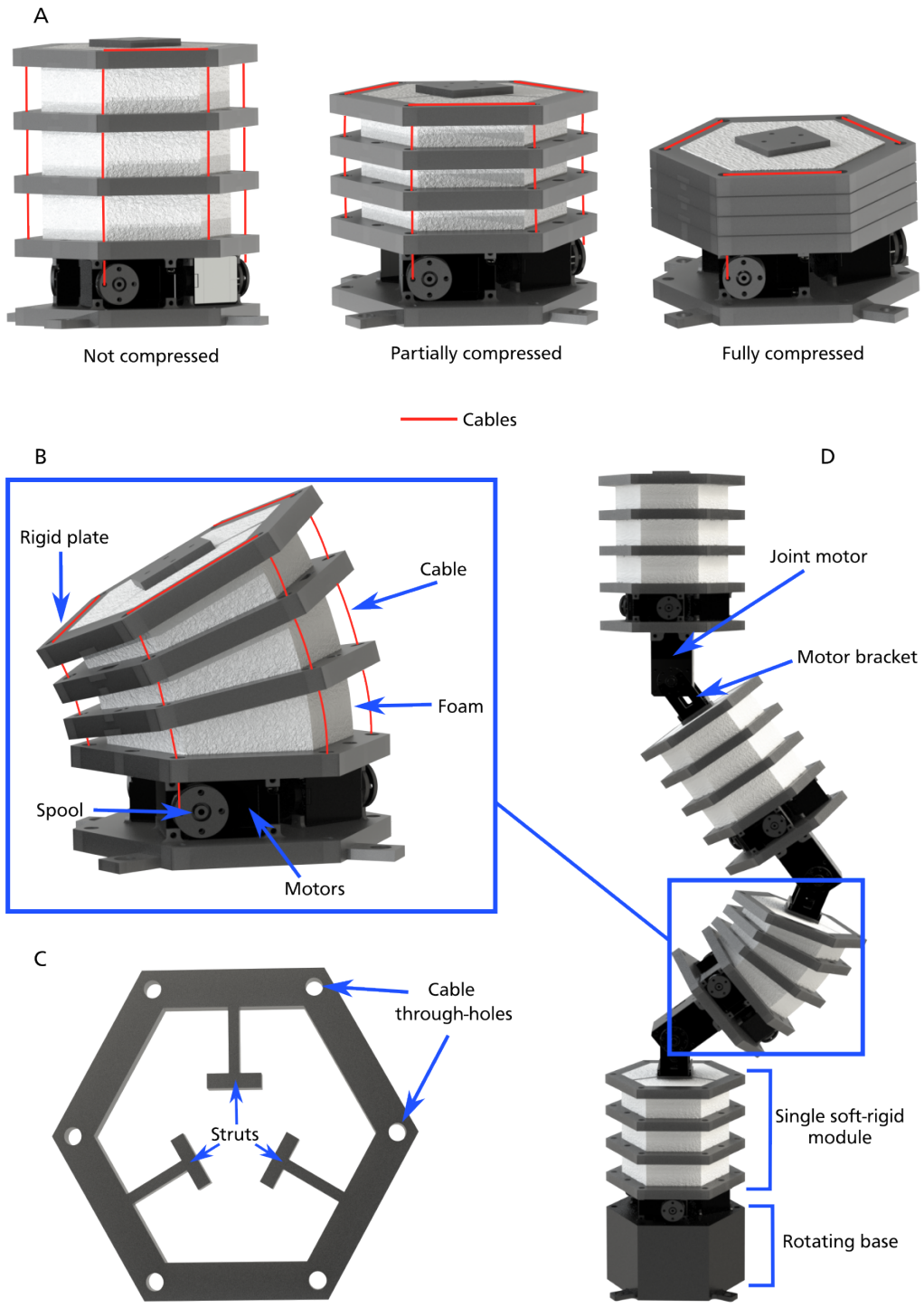


Figure 3.1: Design overview of the soft-rigid manipulator:
(A) Panel visualizing a soft-rigid module transitioning between its soft and rigid states. As the cables contract, the rigid plates are pushed closer together, increasing overall stiffness. **(B)** A labelled soft-rigid module that relies on three dynamixel motors, spools, and cables to achieve various levels of stiffness and shape. **(C)** A single 3D printed hexagon plate, featuring its struts for better foam adhesion and through-holes for cables. **(D)** A full body rendering of the soft-rigid arm, composed of four soft-rigid modules.

3.2.3 Control System

Due to the intrinsic compliance of soft-rigid modules, they can theoretically be defined as having an infinite number of degrees of freedom, which would be computationally expensive to model and integrate within a feedback-controller[38]. To run our control loop in hardware, we rely on the classical Piecewise Constant Curvature (PCC) kinematic assumption from Jones et.al [39], which maps lengths of cables l of a module i to arc length s_i, curvature κ_i and direction of bending ϕ_i.

The equations for each module i are as follows,

$$\kappa_i = \frac{2 \cdot \sqrt{l_1^2 + l_2^2 + l_3^2 - (l_1 \cdot l_2) - (l_2 \cdot l_3) - (l_1 \cdot l_3)}}{d \cdot (l_1 + l_2 + l_3)} \tag{3.1}$$

$$\phi_i = \arctan\left(\frac{\sqrt{3} \cdot (l_3 + l_2 - 2l_1)}{3(l_2 - l_3)}\right) \tag{3.2}$$

$$s_i = \frac{l_1 + l_2 + l_3}{3} \tag{3.3}$$

where d is the distance from the center of the module to the corner of the plate, and l is the length of a module's cable measured from the center of its corresponding motor's horn to the top of the module.

While κ and ϕ have classically been used in soft robot control, it is limited by the singularity that occurs when a continuum segment is completely upright. Therefore, we map this configuration to the one described in [40], which has no singularities or discontinuities, like so:

$$\Delta x_i^2 = \kappa_i d_i \cos(\phi_i) \tag{3.4}$$

$$\Delta y_i^2 = \kappa_i d_i \sin(\phi_i) \tag{3.5}$$

$$\delta L_i = s_i - l_{0,i} \tag{3.6}$$

This allows us to define each module's configuration as $\mathbf{q}_{\text{mod},i} = [\Delta_{x,i}, \Delta_{y,i}, \delta L_i]^T$, where $\Delta_{x,i}$ and $\Delta_{y,i}$ represent the change in length along the x and y axis respectively and δL_i represents the change in arc length. Therefore, the state of the four-module manipulator can be defined as $\mathbf{q} = [\theta_1, \mathbf{q}_{\text{mod},1}, ..., \theta_4, \mathbf{q}_{\text{mod},4}]^T$, where θ refers to the joint motors, which we treat as standard rotational joints. We recognize that since this state representation is calculated solely from the module's cable lengths, there may be slight inaccuracies in a module's state when it is in a softer state. Refer to 5.2, for alternative methods. Still, relying on the motor's encoder information means we don't have to incorporate external sensing into our system. We define the dynamics of SRHA with the following:

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} + G(\mathbf{q}) + K(\mathbf{q}) + D\dot{\mathbf{q}} = A(\mathbf{q})\boldsymbol{\tau} + J^T f_{\text{ext}}, \tag{3.7}$$

where $M(\mathbf{q})$, $C(\mathbf{q}, \dot{\mathbf{q}})$, and $G(\mathbf{q})$ are the inertial, coriolis, and gravitational terms, respectively. They are computed using Featherstone's dynamics algorithm[41]. $K(\mathbf{q})$, D, and $A(\mathbf{q})$ are the stiffness, damping, and input matrices. Finally, $\boldsymbol{\tau}$ is the input vector, J is the end

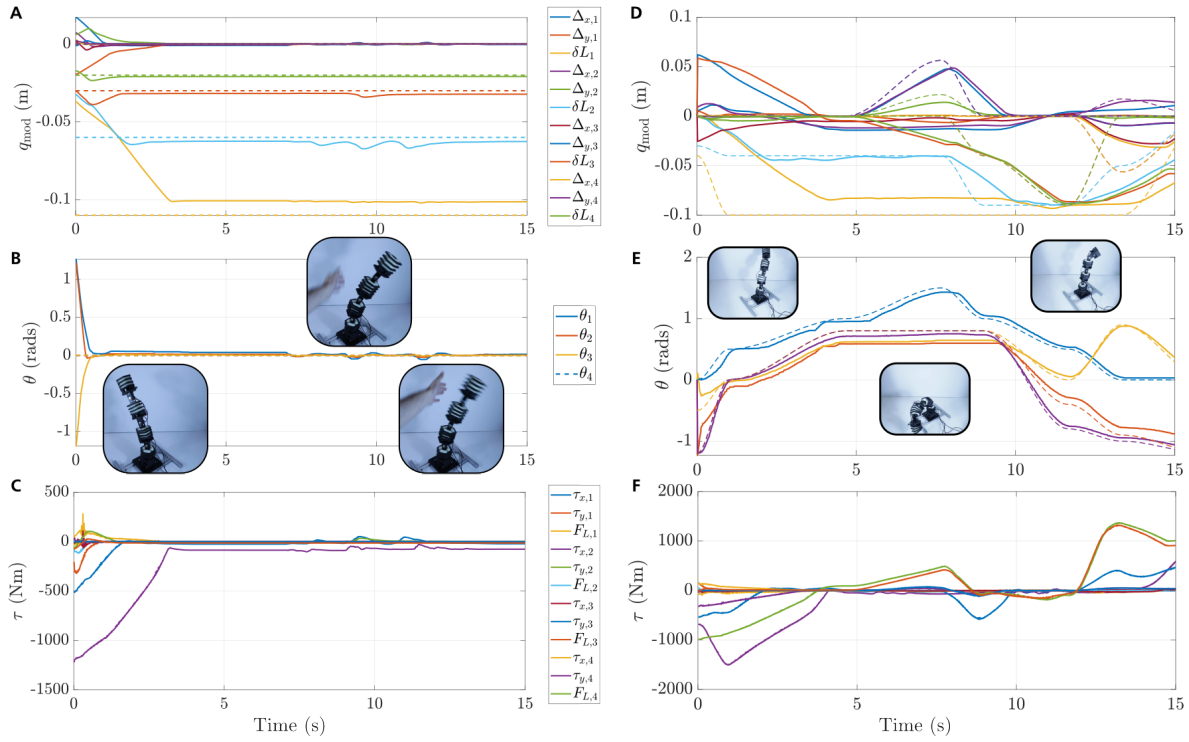


Figure 3.2: **Disturbance Rejection and Trajectory Tracking:** Step response and disturbance rejection for (**A**) modules and (**B**) joints. The inset images are snapshots of the robot as it is perturbed from the set point. (**C**) Torques output by the controller. Trajectory Tracking for (**D**) modules and (**E**) joints. The inset images are snapshots during the trajectory. (**F**) Torques output by the controller.

effector Jacobian, and f_{ext} refers to the external force experienced by the end effector. This then leads to an adapted PD+ controller, adapted from [20], [42], of the following form:

$$\boldsymbol{\tau} = A^{-1}(M\ddot{\mathbf{q}}_{\mathbf{d}} + C\dot{\mathbf{q}} + G(\mathbf{q}) + K\mathbf{q}_{\mathbf{d}} + D\dot{\mathbf{q}}_{\mathbf{d}} + K_{\text{P}}(\mathbf{q}_{\mathbf{d}} - \mathbf{q}) + K_{\text{D}}(\dot{\mathbf{q}}_{\mathbf{d}} - \dot{\mathbf{q}}) + \mathbf{F}_c), \qquad (3.8)$$

This adapted version handles the frequent self contact between adjacent rigid plates of the soft-rigid modules by relying on the contact compensation term, $\mathbf{F}_c$, which relies on a sigmoid function that saturates as the distance c between plates reaches 0. The necessity for a contact compensation term is a practical one. Implementing the controller without contact compensation would still decently work. However, it subjects the modules to unnecessary amounts of force between its plates, leading to faster deterioration of cables and parts. The compensation term simultaneously ensures more efficient use of current passed into the motors as well as an increase in usage of the manipulator's modules.

To deploy this controller on hardware, we set all Dynamixels to current control mode, and we map the torque output of the controller to motor currents using a linear approximation. For faster operation, the Featherstone algorithm used to calculate the manipulator's dynamics, along with the general controller are converted into a C library using MATLAB's Code Generation Toolbox. An overview of trajectory tracking results can be found in 3.2.

3.3 Characterization

We characterize the stiffness range of a single soft-rigid module by measuring the vertical displacement experienced by a module as it is increasingly compressed, with increasing weight suspended at its end-effector. For each trial, the module is mounted on a stable bench top and the cables are contracted to a predefined length. We then suspend a weight to the end of the module and measure the vertical displacement, Δh. We use general beam theory to calculate stiffness $k = \frac{mg}{\Delta h}$, where mg is the gravitational force experienced by the module. For this experiment, we tested five cable lengths and three weights. To reduce memory effect of the foam, we let the module fully decompress for 3 minutes in between trials. As seen in the figure A of 3.3, we at first see a general linear trend between cable length and stiffness, until the point at which stiffness dramatically increases. This jump in stiffness is approximately 30 times the stiffness of the module at its neutral state. It occurs at the point when the cables have pushed the module's rigid plates to the point of complete contact, making the module a rigid body. This confirms the notion that the stiffness of our soft-rigid modules is coupled with the shape. When the cables are fully retracted, the module can bend freely. However, increasing the stiffness directly correlates to a smaller distance between the plates, meaning that a stiffer module can achieve less curvature due to the distance between its plates being smaller. Next, we characterize the workspace of SRHA in simulation by relying on the forward kinematics of its end-effector. This involves passing in a series of random robot configurations to generate the point cloud seen in B of 3.3. The sparsity seen in the middle of the point cloud is due to the limited number of configurations with the end-effector falling into this regime. We also qualitatively characterize the manipulator's load-bearing ability as seen in C of 3.3, where a weight of 300g is suspended from the top most module of SRHA when it is in a completely rigid and compliant state. When all of its modules are fully compressed the manipulator only slightly deviates from its original position. Whereas the manipulator in its completely uncompressed state collapses due to the weight.

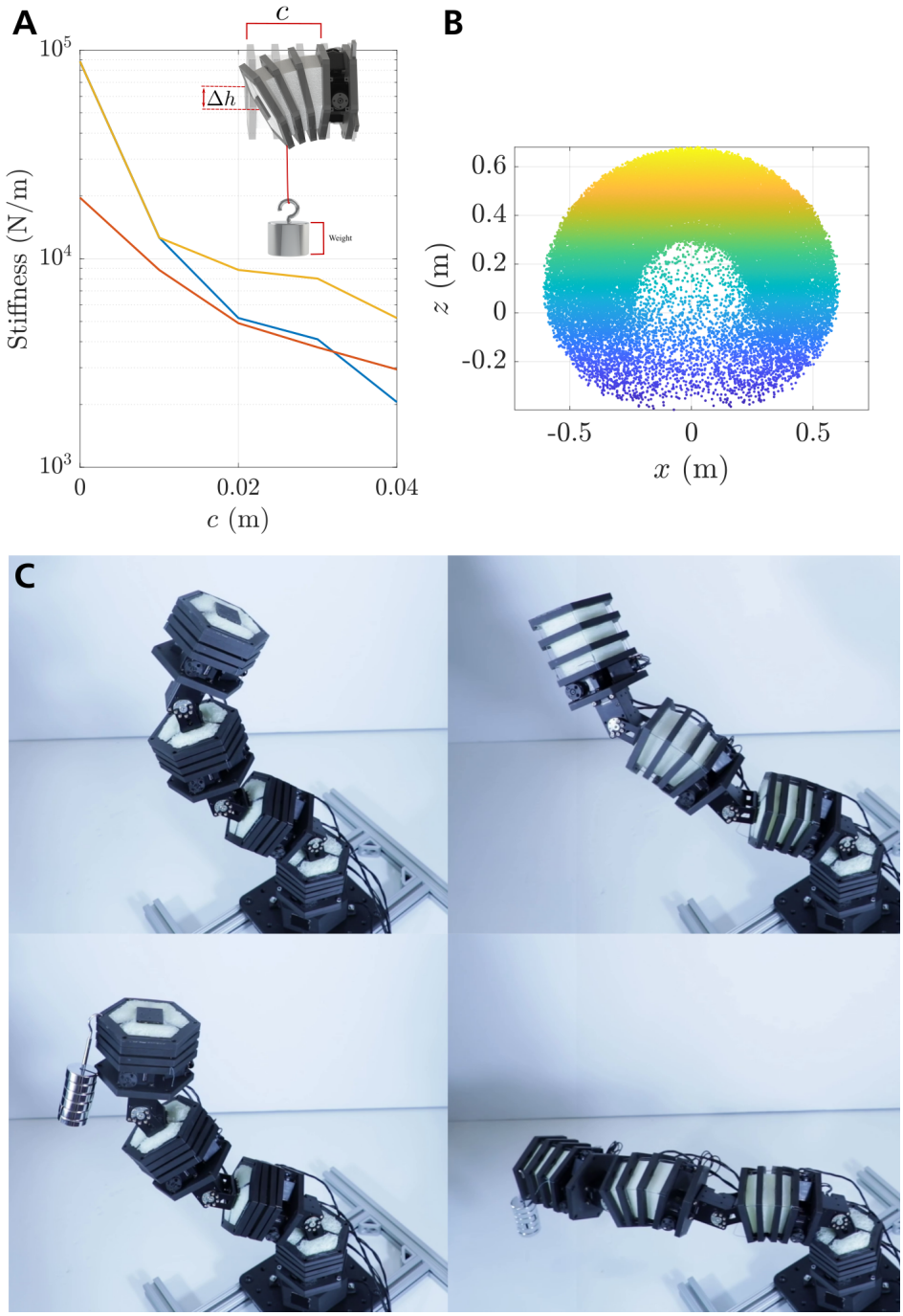


Figure 3.3: **Characterization of Soft-Rigid Module:** (**A**) Plot of the bending stiffness test results. Inset is a rendering of the test performed on a module of the soft-rigid arm. (**B**) Planar slice of the manipulator workspace (which is radially symmetric). (**C**) Demonstration of manipulator supporting a 300 gram mass while the modules are in a rigid state (left) and a flexible state (right).

3.4 Experiments

We demonstrate the manipulator's maneuverability and load-bearing capacities through two object recovery experiments inspired by common human actions: reaching behind the back and bending down to retrieve an object. These actions necessitate bending, extension, and reorientation of the manipulator. Both experiments use the same two plates of 30.5x61 mm acrylic held up by several bars of 8020 aluminum. Each plate is laser cut with four holes of 152mm in diameter. The two acrylic plates are aligned at a 90 degree angle from each other. We also attach a 31.5mm long hook onto the end of the fourth soft-rigid module of the arm to grant the arm the ability to pick up objects. It should be noted that all trajectories carried out in these experiments are manually pre-defined by the user. This is done by defining a series of states with corresponding time points to generate a more comprehensive trajectory using MATLAB's cubicpolytraj function. This gives us a list of desired states, velocities and accelerations that we pass into our controller. For both trajectories we set the first state of all four modules to be completely soft ($dL = 0$), and we set the motor joints to predefined values so that the manipulator rests on the tabletop at the beginning of each experiment.

3.4.1 Red Mug Task

For the red mug experiment, we position a metal red mug from the YCB dataset [43] behind the bottom right circular opening of the left-most acrylic plate, as shown in Figure 3.4. We align the mug with its handle protruding through the opening to assess the soft-rigid arm's ability to reorient and essentially "reach behind" itself, akin to how a person can typically reach behind their back to grasp objects.

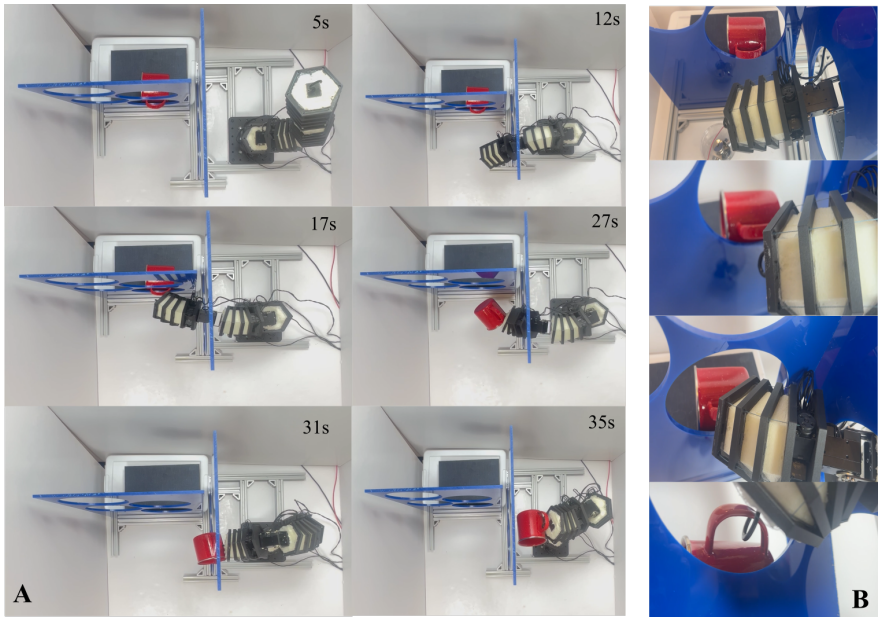


Figure 3.4: **Experimental Results of Red mug Task: (A)** Left panel features snapshots of the soft-rigid manipulator bending its modules in order to grab the red mug. **(B)** Right panel is a close up of SRHA's end-effector module reorienting its hook to retrive the mug.

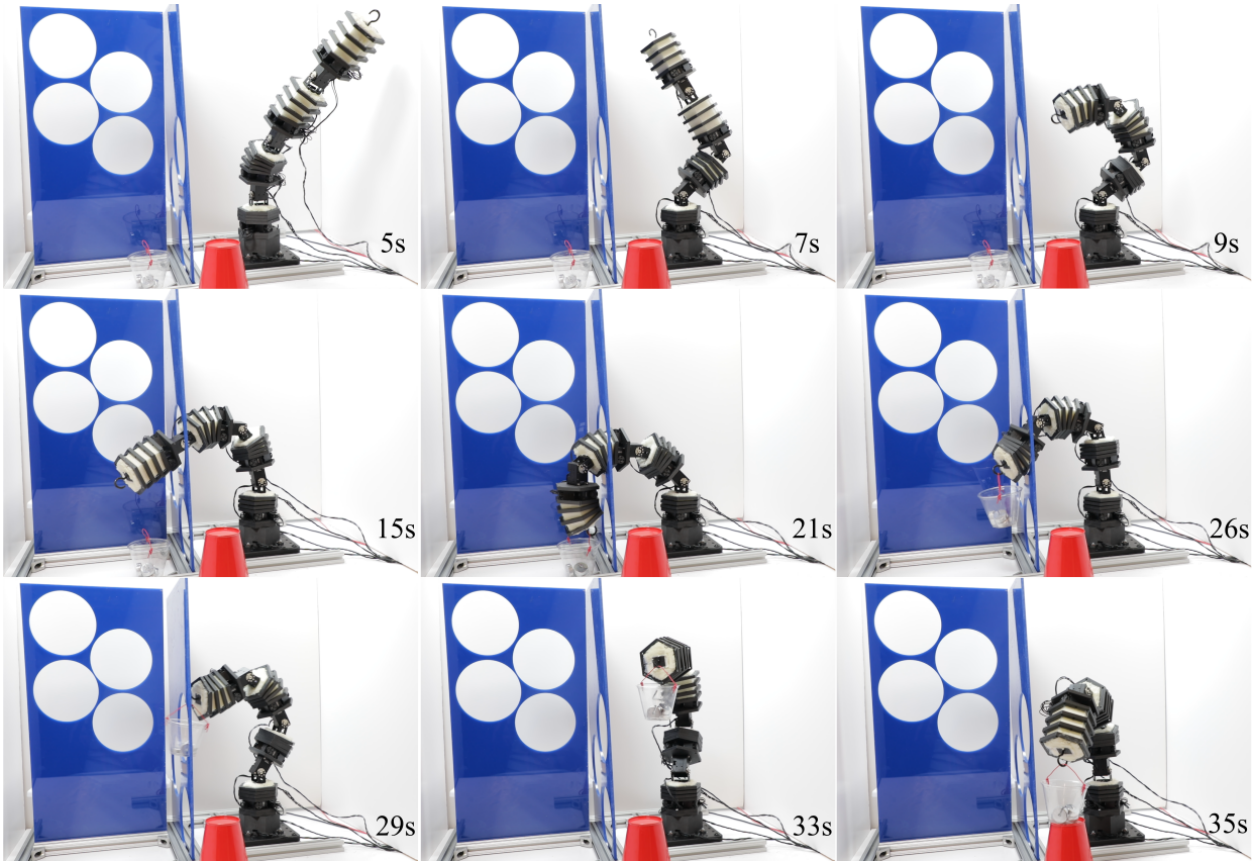


Figure 3.5: **Experimental Results of Weighted Cup Task:** Features the soft-rigid manipulator going through the obstacle course to pick up a cup with around 200 grams of lab weights.

3.4.2 Weighted Cup Task

For the weighted cup experiment, we place four lab weights, totaling 200g, into a plastic cup with an attached handle. Deliberately, we position the weighted cup closer to the wall of the right-most acrylic plate. This setup challenges the manipulator to bend almost to the surface of the tabletop, as seen at the 21 second mark of 3.5 while also demanding enough strength to successfully retrieve the cup of weights, showcasing SRHA's ability to bend down and lift itself to nearly full height while handling load. Additionally, we test precision by requiring SRHA to accurately place the weighted cup onto a red platform at a predetermined location within the manipulator's workspace. In the same figure, we observe SRHA adapting to the additional 200g load by stiffening certain body parts, yet still maintaining sufficient flexibility and precision to navigate through the obstacle course and deliver the payload accurately.

3.5 Empirical Analysis

While a conventional rigid manipulator would easily be able to pick up both the red mug and weighted cup, it would struggle to reconfigure itself through the obstacle course due to its limited number of links. In contrast, a soft robot, with infinite degrees of freedom, would efficiently maneuver through the circular openings but lack the strength and precision needed to retrieve the objects. The soft-rigid manipulator leverages a combination of stiffening and bending to achieve both precision and flexibility, effectively merging the strengths of both rigid and soft systems.

For the first phase of both experiments we rely on the full compression of the bottom module. This module's compression is crucial for providing the necessary stability and strength to support the arm and the attached load.

Next, we engage the second and third modules to bend, providing the flexibility and range of motion needed to maneuver through the complex environment. Their bending capabilities are essential for adjusting the arm's orientation and extending or shortening certain parts of its body to reach the goal position in 3D space. In the case of the red mug, the second and third modules bent in the direction of the red mug to provide better reach for the fourth module. In the case of the weighted cup experiment, additional compression is needed at the second module, as seen at the 26 second mark 3.5, when the manipulator began elevating the cup.

For the red mug experiment, we slightly bend and compress the fourth module to achieve precise control over the manipulator's hook. This precision is necessary for the end effector to hook onto the handle of the mug securely, as seen in B of 3.4. With the mug secured, the fourth module maintains its slight compression and bend to ensure maintain grasp while the arm reverses its path and returns to starting home position.

For the weighted cup experiment, more extreme bending moments are required, along with additional assistance from the rigid joint motors, to compensate for the significant weight of the cup on the manipulator. Therefore, when designing the trajectory for this experiment, we introduce slightly more compression on the end-effector module and utilized the joint motor between the second and third modules for additional support. This experiment exemplifies the necessity for both soft and rigid elements to work in collaboration. Without the rigidity provided by the joint motors and the compression of the end-effector module, the manipulator would not have been able to pick up the weighted cup. Additionally, without the compliance from the second and third modules, the manipulator would not have been able to navigate in and out of the course.

The results of these experiments demonstrate SRHA's ability to leverage the unique properties of its soft-rigid modules at various parts of its body to accomplish complex tasks. The successful navigation of the obstacle course and precise handling of the mug and weighted cup highlight the manipulator's potential for applications requiring both load-bearing capacity and fine manipulation in unstructured environments. These findings underscore the effectiveness of our design and control strategies in enhancing the collaboration between the soft and rigid aspects of soft-rigid robotic systems for human-interaction.

Chapter 4

Robotic Sea Turtle with Soft-Rigid Fins: Parallel Configuration

4.1 Motivation

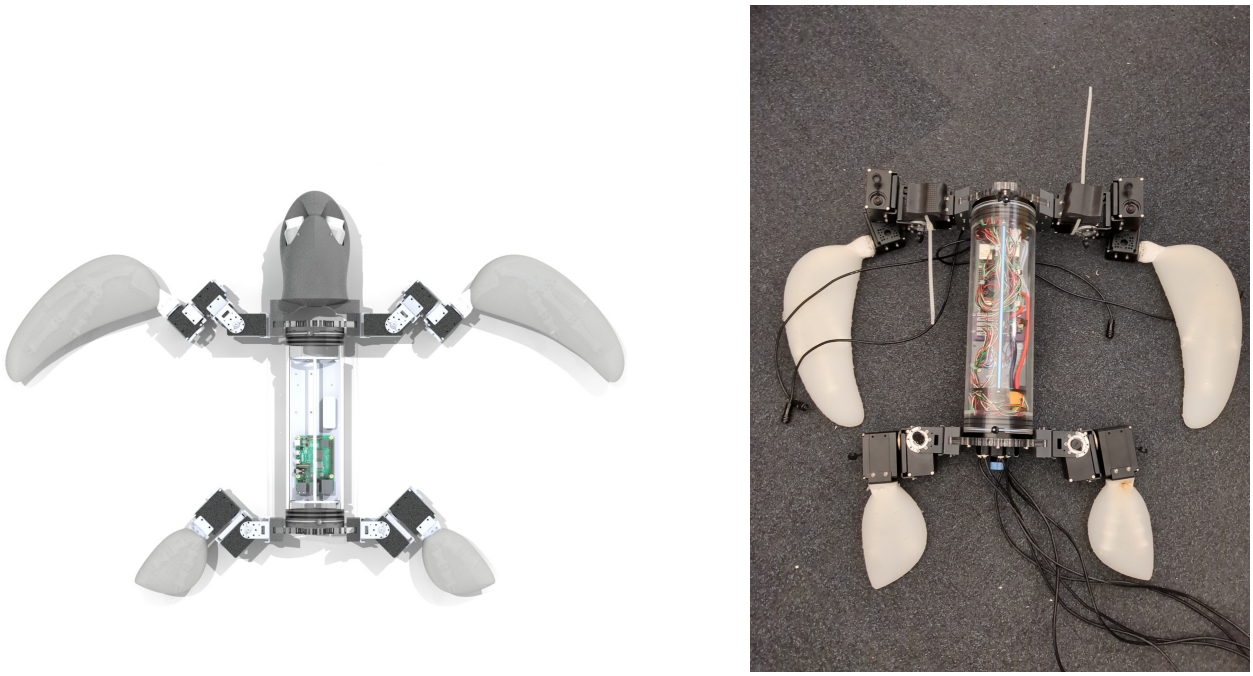
Although various types of underwater vehicles exist, there are still occasions where animals become startled and swim away, potentially due to the appearance of the robot or its noisy propulsion system [44], presenting challenges for conducting long-term studies in the wild. One solution is a bio-inspired robot that looks and behaves like its animal counterpart, making them less invasive from a visual and behavioral standpoint, potentially enabling researchers to conduct closer studies of marine animals in their natural habitat. In this chapter, we employ a robotic sea turtle equipped with flippers inspired by the anatomy of real sea turtle flippers to replicate their swimming gaits. A sea turtle's locomotion involves several key anatomical components. Firstly, the parallel configuration of bones embedded within its compliant flipper provides the necessary support and degree of rigidity essential for moving the flippers with sufficient speed and precision. Secondly, the surrounding soft tissue and muscle covering the skeletal structure enable the flipper to bend and twist, facilitating streamlined movement through the water and thrust through its flapping-like motion[31], [32], [45]. Most underwater simulators that exist in the literature are for conventional AUV or ROVs, and simulating soft robots is a challenging problem in itself [46]. Therefore, to generate locomotive trajectories we consider a data-driven, reinforcement learning approach.

This chapter focuses on applying a policy gradient method to learning a forward swimming gait for the robotic sea turtle in an online manner. First, we introduce the framework, along with how we parameterize the problem to learn CPG parameters rather than a conventional policy network. Next, we validate the approach in simulation with typical benchmark systems. Finally, we deploy this framework on the robotic platform in an underwater setting.

The following thesis work uses an existing robotic sea turtle platform that was developed in the Distributed Robotics Laboratory.

4.2 System Overview

The main features of the robotic sea turtle platform are the BlueRobotics® watertight enclosure that houses all the electronics, the four soft-rigid flippers, and the 10 waterproof motors from Dynamixel that actuate said flippers. The platform propels itself underwater by relying on two soft-rigid front flippers and two smaller soft-rigid back flippers. The front flipper relies on three XW540-T260-R Dynamixel motors that can be identified as the shoulder, middle and flipper motor. They are connected to each other using FR13-H105K brackets. Each back flipper is actuated by two XW540-T140-R Dynamixels and connected the same way as their front counterparts. An overview of the robotic system can be seen in 4.1.



(a) CAD image of turtle robot.

(b) Real life image of turtle robot.

Figure 4.1: The robotic sea turtle with soft-rigid flippers.

The motors and electronics are powered by a 11.1 V LiPo battery. To control the motors, we use a custom power hub board and a small USB communication converter (U2D2) from Dynamixel to connect to the Raspberry Pi 4 Model B mounted within the robot's water-tight enclosure. The robot also has a 9-DOF IMU, an INA219 high side current sensor, and a Bar30 depth/pressure sensor from Blue Robotics. All sensors are read from a Seeed Studio XIAO nRF52840, a low-powered micro-controller.

4.3 Policy Search Framework

4.3.1 Central Pattern Generator

Central Pattern Generators (CPGs) have allowed robot fish to accomplish a variety of motions, such as swimming and crawling by abstracting the motion of their fins. In Crespi [34], each fin on their robotic fish is modelled as a system of three coupled amplitude-controlled phase oscillators as seen in the mathematical description below:

$$\dot{\phi}_i = \omega_i + \sum_j \left(w_{ij} r_j \sin(\phi_j - \phi_i - \varphi_{ij})\right), \tag{4.1}$$

$$\ddot{r}_i = a_r(\frac{a_r}{4}(R_i - r_i) - \dot{r}_i), \tag{4.2}$$

$$\ddot{x}_i = a_x(\frac{a_x}{4}(X_i - x_i) - \dot{x}_i), \tag{4.3}$$

$$\theta_i = x_i + r_i \cos(\phi_i). \tag{4.4}$$

For an oscillator i, the set point is defined as θ_i in radians. Our state variables are ϕ_i, r_i, and x_i which represent the phase, amplitude, and offset of the oscillations (in radians), respectively. The control parameters we would optimize for would be ω_i, R_i, and X_i, which [34] defines as the desired frequency, amplitude, and offset of the oscillations. The parameters w_{ij} and ϕ_{ij} are coupling weights and phase biases which determine how oscillator j influences oscillator i. However, for a robot with a more complicated action space like the sea turtle, a better CPG implementation to look at would be in Bellegarda[47], where they use an uncoupled CPG framework to generate quadruped walking gaits. This is accomplished by simply setting the intrinsic weights to zero. We choose this approach since a quadruped is more similar to the structure of a sea turtle than a fish robot, due to their respective limbs having multiple degrees of freedom and requiring a certain amount of coordination to collectively generate a gait.

For both the robots in simulation (refer to 4.4) and sea turtle robot, each motor i is represented by a single, uncoupled CPG, whose output θ_i corresponds to the position in radians of the motor. Since the goal for the sea turtle robot is to learn a forward swimming motion, inherently there is some symmetry in the way the sea turtle robot should swim. To simplify the learning space, we constrain the amplitude search space to strictly positive values and use a mirrored version of the CPGs for one side of the sea turtle. This approach ensures that, even if the motions are initially out of phase, there will be fewer occurrences of the flippers actuating in opposite directions, which clearly encourage more roll-like behaviors than propulsive ones. An overview of the parameter search space for all robot agents is defined in 4.1. We use the same limits defined by [48] for the simulated agents. The limits for the robot sea turtle were determined by the mechanical motion constraints of the joint motors.

4.3.2 EPHE

Previous work has shown policy gradient methods being very successful in generating useful gaits for robots with continuous observation and action spaces, such as quadrupeds [49]. However, commonly used policy gradient methods, such as Proximal Policy Optimization (PPO) [50], need their agent to at least be partially pre-trained in simulation before being integrated into the physical hardware setup and environment. Currently, we do not have a proper simulator to capture the dynamics of our robotic platform. Therefore, our agent can only be fed real-world data, which calls for a policy gradient method that does not require a large number of samples. For training, we use the EM-based Policy Hyper Parameter Exploration (EPHE) algorithm [51], a policy gradient-based reinforcement learning method aimed at learning the optimal CPG parameters to achieve the largest forward reward. EPHE combines features from PGPE [52] and EM-based Policy Search [53]. It draws from a Gaussian distribution defined by an initial set of priors: μ and σ. For an M number of rollouts, it then updates the priors using the k best CPG parameters, θ, that had the highest rollout return, $R(h)$, where h is a sequence of states, actions, and rewards. This process repeats for several episodes until convergence. In our case, we set a maximum number of episodes for learning. Additionally, we evaluate a random set of parameters within the limits of the agent's action space and select the parameters with the highest reward to define the initial center of distribution, μ. This helps increase the number of positive rewards in the first episode of learning, as the algorithm assumes that $R(h)$ is strictly positive.

Algorithm 1 EPHE algorithm with Parameter Sweep

Run parameter sweep
for $n = 1$ to N **do**
    draw $\theta \sim \mathcal{U}(a, b)$
    evaluate $R(h^n)$
end for
Input: initial μ (CPG parameters from parameter sweep) and initial σ (standard deviation)
for each episode **do**
    for $m = 1$ to M **do**
        Perform trajectory m
        for each trajectory **do**
            draw $\theta_i \sim \mathcal{N}(\mu, I\sigma_i^2)$ for all i
            evaluate $R(h^m)$
        end for
    end for
    Select k best trajectories from the sorted $R(h^m)$
    Update μ and σ
    $\mu = \frac{\sum_{k=1}^{K}[R(h^k)\cdot\theta_i^k]}{\sum_{k=1}^{K} R(h^k)}$
    $\sigma = \sqrt{\frac{\sum_{k=1}^{K}[R(h^k)(\theta_i^k-\mu_i)^2]}{\sum_{k=1}^{K} R(h^k)}}$
end for

4.4 Simulation

To validate EPHE for locomotive robots, we first run the CPG-EPHE framework on the half-cheetah robot in simulation. The Half-Cheetah has 6 joint motors, 3 for each leg. We therefore have 6 CPGs and a total of 13 parameters to learn from. The 13th parameter refers to the frequency, which we apply to all CPGs. To map CPG position outputs to torque, we rely on a simple PD controller from [48]. We run 30 trials of the half-cheetah robot simulation, where our priors μ and σ are set similarly to that in [51]. We set the intrinsic weights of the Half-Cheetahs CPGs a_r and a_x to 20. For the EPHE algorithm, we set M to 20, k to 10, and allow the algorithm to run for a maximum of 10 episodes. We rely on Mujoco's predefined reward function for the Half-Cheetah, which is comprised of a forward reward defined by the positive distance the Half-Cheetah makes in a time step, and a cost that penalizes the cheetah for taking large actions. Since the EPHE algorithm relies on rewards being strictly positive, we set σ and M to be high enough such the during the first episode the agent explores enough to generate a few positive rewards. If one of the k best rewards happens to be negative, we automatically clip it to 0. While this works effectively for the Half-Cheetah robot, as seen in the graph 4.2a, the same approach fails for the Ant robot. This most likely is due to the fact that the Ant agent does not operate in a planar environment and has a larger action space, with four legs instead of two. Since each leg has two motor joints, that maps to 8 CPGs and therefore a total of 17 parameters. Furthermore, operation within the 3D space allows the agent to explore walking in multiple directions (left, right, diagonally), whereas the half cheetah is restricted to only backwards and forward motions. Additionally, the Ant's reward function is defined by more terms, such as a health and contact penalty to prevent extreme actions. Therefore, it requires a larger number of rollouts to increase the chances of acquiring a positive reward for the Ant environment. However, if we make M very large, it decreases the sample-efficiency. We take a different approach similar to [49], where we first pass in a series of N random parameters (refer to 4.3.2) before running EPHE and set μ to be the parameters that generated the largest amount of reward, even if that reward was negative. As seen on the right graph 4.2b, the ant agent is able to learn with parameter sweeping, without the need of increasing the number of rollouts, M. We apply this parameter sweeping method to the robotic sea turtle, which has 10 motors mapped to 10 CPGs. This means we need to learn 21 parameters, which is double the number required for the Half-Cheetah and almost double that of the Ant environment.

Environment	**Frequency** ω	**Amplitude** r_i	**Offset** x_i
Ant-v4	$2\pi \cdot U(0.4, 2)$	$U(-1, 1)$	$U(-1, 1)$
HalfCheetah-v4	$2\pi \cdot U(0.4, 5)$	$U(-2, 2)$	$U(-1, 1)$
Robotic Sea Turtle	$2\pi \cdot U(0.4, 5)$	$U(0, 2)$	$2\pi \cdot U(0, 1)$

Table 4.1: Table of Parameters for Simulation and Real-life Agents

4.5 Deployment on Hardware

Due to the difficulties involved in simulating underwater dynamics and deformation of soft materials, such as our robot's elastic flippers, we are unable to efficiently perform RL training in simulation. Therefore, we implement our RL algorithm directly onto our system. We define our forward reward function with the following terms:

- Acceleration penalty, body x direction: a_x^2
- Acceleration reward, body y direction: a_y^2
- Acceleration penalty, body z direction: a_z^2
- Tau penalty: $-\|\tau\|^2$
- Quaternion penalty: $1 - \left|q_g^\top q\right|$

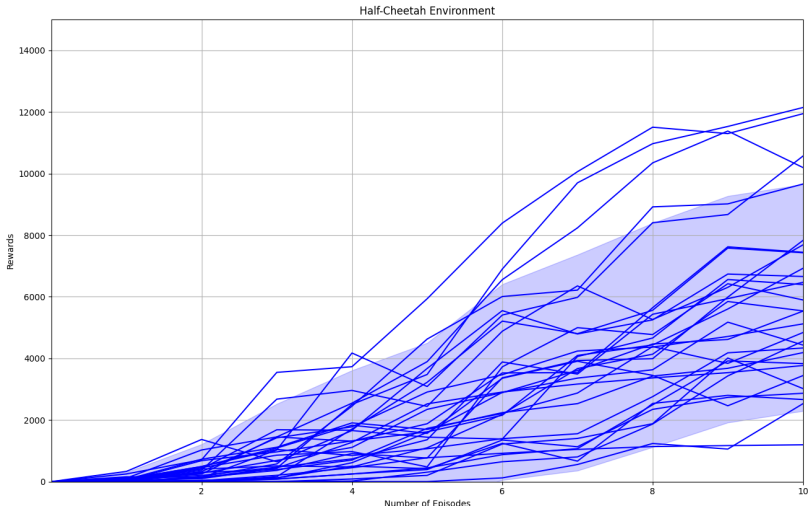
The robot's forward orientation corresponds with the y-axis of its accelerometer, which is why we reward any accelerations along a_y and discourage the other two axes. We get the quaternion penalty from [54], which penalizes the geodesic distance between the goal (q_g) and measured (q) quaternions. We calibrate this value at the start of each hardware experiment for learning.

We still initially try setting similar priors for the robotic sea turtle as what was done for the Half-Cheetah. However, the large action space of the robot was too large to reliably generate a positive reward within the first episode. To counteract this, we implement several adaptations. Firstly, we limit the range of amplitudes for the agent to learn on to be strictly positive, as seen in 4.1, since a forward trajectory has inherent symmetry. We induce this by mirroring the CPG output for the right half of the robot, which moves in the opposite direction of its left counterpart when a positive current is passed in all the motors. Secondly, before learning, we first pass in a series of random CPG parameters, as was done with the Ant agent, and take the one with the highest reward, even if it's still negative. WE further simplify the parameter sweeping range by allowing a large range for frequency, amplitudes, and offsets for the front flipper motors and a smaller range for the back flippers. For the pool test, we set M to 20, k to 5, and run for a maximum of 5 episodes. For the sea turtle robot to learn within a considerable time window, we set the duration of each rollout to 7 seconds to allow enough time for the robot to locomote its flippers several times.

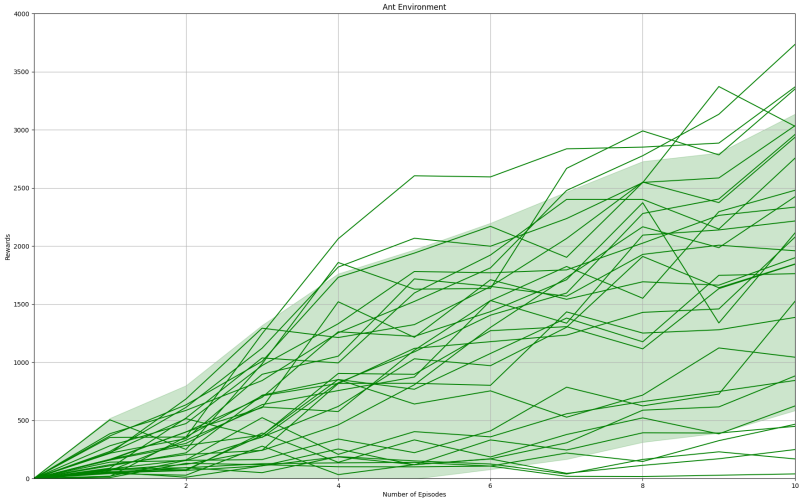
4.6 Results

We observe in the left panel of 4.2a a general trend of learning across the 30 trials of EPHE implemented within the Half-Cheetah simulation, with noticeable variability in rewards. This variation in learning curves is due to the inherent stochasticity of the algorithm, where the learning can be heavily dictated by what the agent learns in the first episode. This could potentially be counteracted by tuning several parameters within the EPHE algorithm, as is discussed in [51]. We could also implement a similar sweeping method of the parameters before learning, so that the EPHE algorithm can start learning from a favorable set of

parameters from the beginning. On the other hand, 4.2b, we also see large variability in learning a walking gait for the Ant simulation, even though we did use the parameter sweeping method previously discussed in 4.4. This also most likely has to do with the stochasticity and larger number of CPG parameters the EPHE has to learn from.



(a) EPHE results of Half-Cheetah robot.



(b) EPHE results of Ant robot.

Figure 4.2: Learning curves of EPHE for optimal CPG parameters.

In the preliminary results for the robotic sea turtle,4.4, we notice a similar trend. We initially see a negative reward during the first episode due to the majority of rewards being negative 4.5. Though the learning framework initially had few positive rewards to learn from, by the second episode we see a jump in rewards. As can be seen in 4.3, the robot sea turtle was successfully able to generate a gait that generally moves forward. The first snap shot at the beginning shows the robot orienting its flippers perpendicular to its body. The robot then generates thrust by the 0.6 second mark by rolling its flippers back, relying on the force between the water and surface of the silicone flipper to propel itself forward. It's worth noting that there remains some asymmetry in this swimming gait. While in a simulation

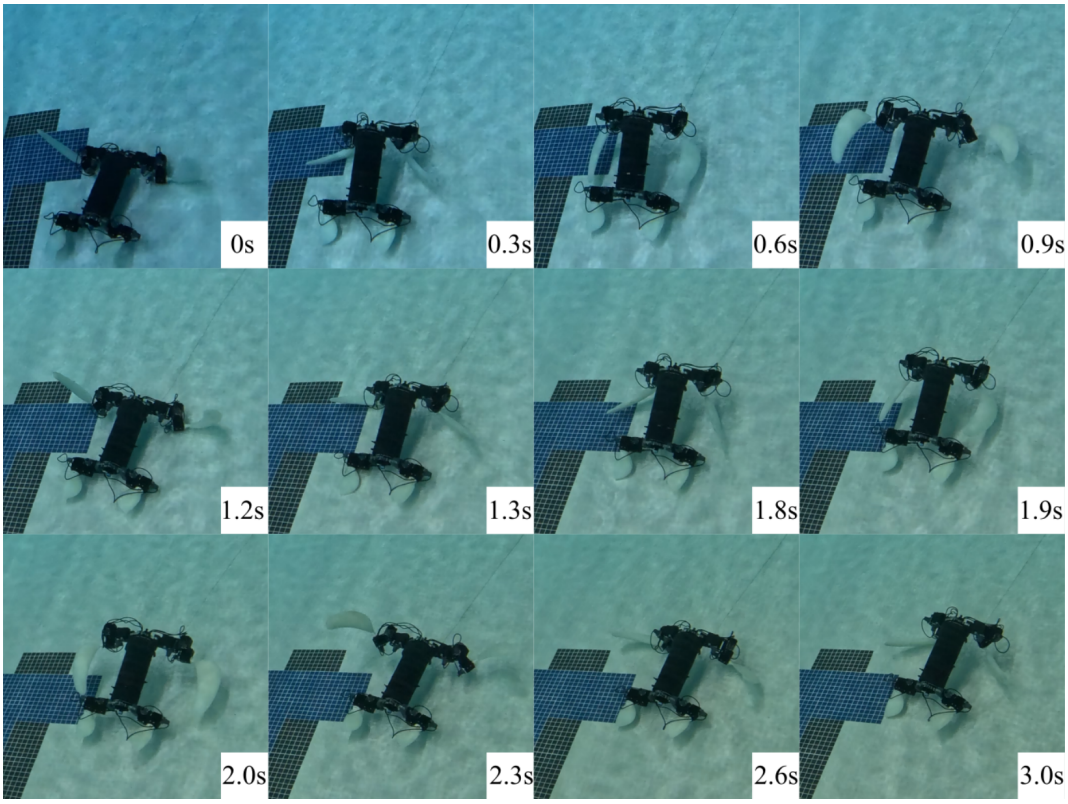


Figure 4.3: Preliminary results of EPHE algorithm deployed on the robotic sea turtle in a pool test setting.

environment, we can reset all state variables and conditions, this isn't feasible for the robot in a pool setting. Due to this limitation, coupled with the brevity of our rollouts (only 7 seconds long), the robot may still achieve substantial rewards from certain CPG parameters that induce circular swimming patterns, for example. This phenomenon has been observed in previous hardware runs, where varying degrees of turning motions resulted in the sea turtle exhibiting circular swimming behavior. Hence, by the 3-second mark, we notice the turtle gradually inclining towards its right side. We also observe that the sea turtle robot tends to converge to a lower depth rather than remaining at the surface. Presently, depth is not factored into the reward function. Consequently, the agent can converge on trajectories where it dives deeper into the pool, incurring penalties due to deviations from the desired orientation. However, the forward acceleration gained during these deeper dives is sufficient to offset any accrued penalties. This could explain why there is a dip in rewards during the fourth episode 4.4, when the robot learns a faster gait and therefore acceleration readings from the other two axes accumulate. This episode could also be the point when the turtle starts turning more towards its right side, collecting larger quaternion penalties that impact the overall episode returns. Further tuning is required on the reward function to mitigate this kind of behavior, as well as tuning of the EPHE parameters.

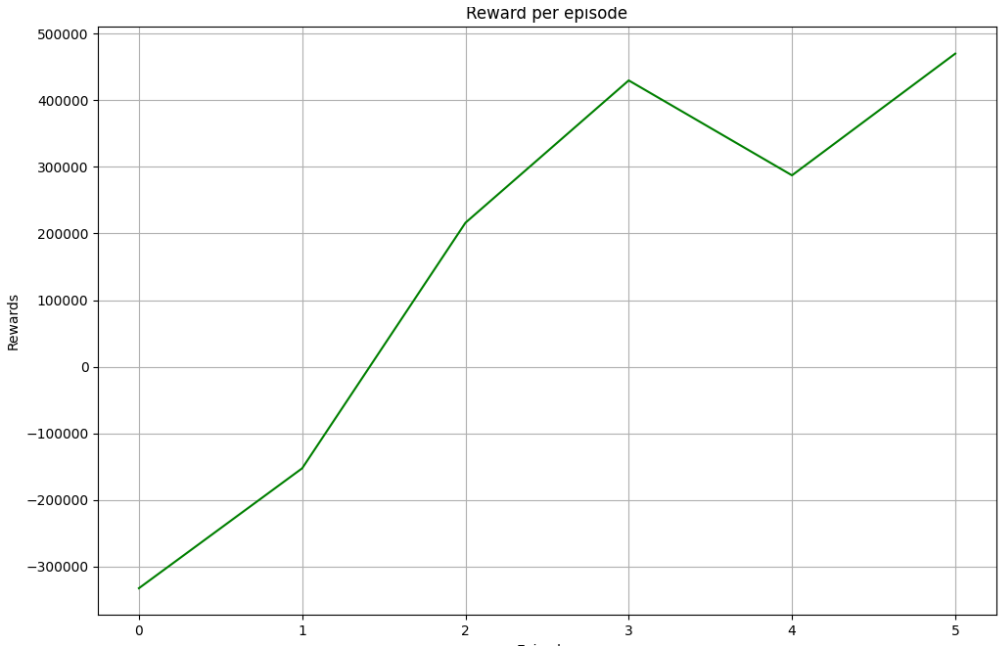


Figure 4.4: Preliminary results of EPHE algorithm deployed on the robotic sea turtle in a pool test setting.

Chapter 5

Conclusions

This thesis investigates how to leverage soft-rigid structures for better interaction and underwater locomotion for robots. To take advantage of the material and structural properties of soft-rigid configurations, we look at both model-based and model-free approaches.

We first consider soft-rigid modules in a series configuration, which we present in a soft-rigid hybrid arm (SRHA) in Chapter 3. This soft-rigid manipulator relies on a model-based control approach adapted from a soft robot PD controller. Due to the unique design of the modules, whose rigid plates make contact, we add a rigid plate compensation factor to account for this interaction, as well as to mitigate unecessary torque output. We also present several characterization and controller experiments, demonstrating the stiffness modulation of soft-rigid modules, as well as their ability, when fully assembled into a hybrid manipulator, to traverse an unstructured environment while also handling load.

We then introduce in Chapter 4 a model-free approach to control the motion of a robotic sea turtle with soft-rigid flippers in a parallel configuration. We approach this by using a CPG-coupled Reinforcement Learning framework that learns on CPG parameters to generate sea-turtle-like swimming gaits without having direct knowledge of the soft-rigid flippers' dynamics. We prove in simulation this RL-CPG framework's ability to aid locomotive agents and present preliminary hardware experiments demonstrating the robotic sea turtle's capability to perform temporary forward swimming gaits.

In conclusion, this research demonstrates the potential of integrating soft-rigid structures in robotic systems to enhance their adaptability and performance in diverse environments. By exploring both model-based and model-free approaches, we provide comprehensive insights into the design and control of soft-rigid robots. Our findings contribute to the advancement of robotics, particularly in applications requiring flexible and efficient movement in complex and unstructured environments, such as a home or underwater environment. Future work can further refine these approaches and extend their applicability to a broader range of robotic platforms.

5.1 Lessons Learned

There are several lessons that can be taken from both robotic platforms.

1. **Importance of Precise Calibration:** Accurate calibration of the joint motors and

sensors is crucial for the successful operation of the soft-rigid hybrid arm and robotic sea turtle. Ensure that all components are calibrated correctly to avoid discrepancies in the movement and control of the robot.

2. **Effective Use of Simulation:** Simulation is your friend. Use it to the best of its ability before physical testing. They can help predict an algorithm's limits and identify potential issues, saving time and resources in the long run. That being said, simulation does not 100 percent guarantee that your framework will smoothly perform in the real-world, especially for systems that incorporate highly non-linear structures like foam (SRHA) and silicone (robot sea turtle).

3. **Iterative Design and Testing:** The Soft-Rigid Hybrid Arm was not built in a day. Its initial conception started with a single module idea drawn on a piece of paper, followed by an initial design in CAD, and then fabricated into a tangible prototype. This prototype still needed to be characterized, integrated with electronics and software, and then pushed to its limits. Design, test, and refine until your list of criteria is met. This leads to incremental improvements, more requirements that were not initially thought of, and therefore a more robust and reliable system at the end of the day.

4. **Collaboration and Communication:** Collaborate with your fellow lab mates! The great thing about collaboration is that each party involved has some knowledge and experience to contribute, meaning you both learn and gain from it. Share your research thoughts and ideas, ask as many questions as you can, and seek out help when you're stuck. Sometimes all you need is a different perspective.

5.2 Future Work

For the next design of SRHA, we intend to move all motors and routing cables to the bottom of the manipulator to expand the manipulator's load-bearing capacity. We are also looking to further leverage stiffness modulation in planning, similar to what has been done in [55]. A first step towards that would be to incorporate better sensing into the system. Instead of estimating the lengths of the cables, directly measuring the distance between the plastic plates could give a better measurement of the robot's state. Time-of-flight (Tof) sensors rely on infrared light to measure the distance between its receiver and target. Embedding three of these compact sensors spaced equally around the bottom plate of each module will measure the relative distance between the Tof plate and plate above it. This improved sensing will enable more effective control and planning.

Additionally, we rely on a more simplified robot to further explore how to handle frequent self-contact of rigid plates in [37] as seen in 5.1 , which relies on Control Barrier Functions (CBFs) to provide guarantees in the smooth operation of the modules during moments of self-contact. A next step would be to adapt this new control approach for SRHA, as well as how we may generally apply CBFs towards regulating smooth operation of non-prehensile tasks for soft-rigid robots.

For the robotic sea turtle, the EPHE algorithm is very limited, and its performance is highly dependent on the quality of the reward function and the initial priors passed in. For

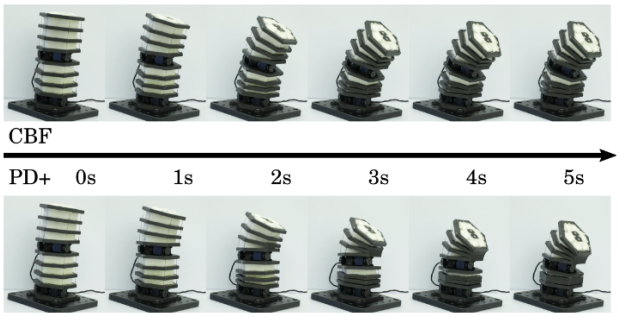


Figure 5.1: CBF vs. PD+ controller for soft-rigid modules dealing with self-contact.

the future, a first adjustment will include integrating a depth sensor into the reward function, as well as potentially incorporating an angular velocity penalty, as is done in [49]. We will also look at applying more powerful learning algorithms to learn a policy generalizable to any kind of swimming gait, not simply for forward motion. Additionally, the current robotic sea turtle design has no sensing within the flippers; their states solely rely on the angles measured from their corresponding joint motors, therefore there is no knowledge of the flippers' deformation. To enable better control and learning, we could embed encoders within the joints of the flipper bones to acquire some knowledge of the flipper's state when actuated underwater, which would allow us to explore force control approaches.

Appendix A

Code listing

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import print_function
import os
import math
from math import cos, sin
from datetime import datetime
import numpy as np
from matplotlib import pyplot as plt
from dynamixel_sdk import *                    # Uses Dynamixel SDK
    library
from Dynamixel import *                        # Dynamixel motor
    class
from dyn_functions import *                    # Dynamixel support
    functions
from FourModController import *                # Controller
from Constants import *                        # File of constant
    variables
from Mod import *
import json
import traceback
from utilities import *
from ctypes import *
if os.name == 'nt':
    import msvcrt

    def getch():
        return msvcrt.getch().decode()

    def kbhit():
        return msvcrt.kbhit()
else:
    import termios
```

```
    import fcntl
    import sys
    import os
    from select import select
    fd = sys.stdin.fileno()
    old_term = termios.tcgetattr(fd)
    new_term = termios.tcgetattr(fd)

    def getch():
        new_term[3] = (new_term[3] & ~termios.ICANON & ~termios.ECHO)
        termios.tcsetattr(fd, termios.TCSANOW, new_term)
        try:
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_term)
        return ch

    def kbhit():
        new_term[3] = (new_term[3] & ~(termios.ICANON | termios.ECHO)
           )
        termios.tcsetattr(fd, termios.TCSANOW, new_term)
        try:
            dr, dw, de = select([sys.stdin], [], [], 0)
            if dr != []:
                return 1
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_term)
            sys.stdout.flush()

        return 0


os.system('sudo /home/zach/SRRA/latency_write.sh')

ctrl = CDLL("puppet_controller_4_cg/puppet_controller_4_cg.so")
puppet_controller_c = ctrl.puppet_controller_4_cg
puppet_controller_c.restype = None
puppet_controller_c.argtypes = np_mat_type(16), np_mat_type(16),
   np_mat_type(16), np_mat_type(16), np_mat_type(16), \
    c_double, c_double, c_double, c_double, c_double, c_double,
       c_double, c_double, c_double, c_double, c_double, \
    np_mat_type(256), np_mat_type(
        256), c_double, c_double, c_double, c_double, c_double,
           c_double, np_mat_type(16), np_mat_type(4)

# Open module port
if portHandlerMod.openPort():
```

```
    print("Succeeded to open the port")
else:
    print("Failed to open the port")
    print("Press any key to terminate...")
    getch()
    quit()

# Set port baudrate
if portHandlerMod.setBaudRate(BAUDRATE):
    print("Succeeded to change the baudrate")
else:
    print("Failed to change the baudrate")
    print("Press any key to terminate...")
    getch()
    quit()

# open big motors port
# Open joint port
if portHandlerJoint.openPort():
    print("Succeeded to open the port")
else:
    print("Failed to open the port")
    print("Press any key to terminate...")
    getch()
    quit()

# Set port baudrate
if portHandlerJoint.setBaudRate(BAUDRATE):
    print("Succeeded to change the baudrate")
else:
    print("Failed to change the baudrate")
    print("Press any key to terminate...")
    getch()
    quit()

packetHandlerMod = PacketHandler(PROTOCOL_VERSION)
packetHandlerJoint = PacketHandler(PROTOCOL_VERSION)

Arm = Mod(packetHandlerMod, portHandlerMod, [
          1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])

Arm.set_max_velocity(MAX_PROFILE_VELOCITY)
Arm.set_current_cntrl_mode()
Arm.enable_torque()

# Base Motor, Joint1, Joint2
Joints = Mod(packetHandlerJoint, portHandlerJoint, [0, 1, 2, 3])
```

```
Joints.set_current_cntrl_mode()
Joints.enable_torque()
# the motor angles of the 9 motors at neutral position
t_old = time.time()
joint_th0 = Joints.get_position()
print("[DEBUG] dt:", (time.time() - t_old))

th0 = [[5.131165735477469, 3.6616121406830255, 3.796602450016962],
       [4.391786995716591, 2.8286605728611223, 5.29990362214489],
       [0.2975922728498144, 4.8182336547487985, 0.5399612373357456],
       [4.7047190764452615, 3.680019910137653, 4.46235011195933O5]]
offset1 = 2.8455343615278643
offset2 = 0
offset2 = 2.208932334555323
offset3 = 2.825592611285351
base_offset = 4.17
print(f"Offset for XM430s: {base_offset, offset1, offset2, offset3}")
BigMotors = [joint_th0[0] - base_offset, joint_th0[1] -
             offset1, joint_th0[2] - offset2, joint_th0[3] - offset3]
print(f"Mod 1 Reference Angles: {th1_0}\n")
print(f"Mod 2 Reference Angles: {th2_0}\n")
print(f"Mod 3 Reference Angles: {th3_0}\n")
print(f"Mod 4 Reference Angles: {th4_0}\n")
print(f"Current theta readings: {Arm.get_position()}\n")
print(f"Z Motor Position: {BigMotors[0]}")
# our max arc length (in m)
s = (l4_0[0] + l4_0[1] + l4_0[2])/3
# cables of the three mods at neutral position
l0 = [l1_0, l2_0, l3_0, l4_0]
# grab cable lengths of all 9 motors
l = [[0, 0, 0], [0, 0, 0], [0, 0, 0], [0, 0, 0]]
l = grab_arm_cable_lens(Arm.get_position(), l, l0, th0, r)
print(f"Cable lengths based off of dtheta: {l}\n")
print(f" big motors angles: {BigMotors}")
q = grab_arm_q(l[0], l[1], l[2], l[3], BigMotors[0],
               BigMotors[1], BigMotors[2], BigMotors[3], s, d)
nq = 16
nmod = 4
nm = 4
q_data = np.zeros((nq, 10))
print(f"Q DATA SIZE: {q_data.shape}")
tau_data = np.zeros((nq, 1))
timestamps = np.zeros((1, 1))
c_data = np.zeros((nmod, 1))
dt_loop = np.zeros((1, 1))
# Report our initial configuration
print(f"Our current q: {q}\n")
```

```
first_time = True
np.set_printoptions(precision=3)
np.set_printoptions(suppress=True)
try:
    while 1:
        print(
            "\nT: Trajectory, W: Set point, C: CALIBRATE FIRST! (or
               press SPACE to quit!)")
        key_input = getch()
        if key_input == chr(SPACE_ASCII_VALUE):
            print("we're quitting\n")
            break
        elif key_input == chr(CKEY_ASCII_VALUE):
            th0 = Arm.get_position()
            th1_0 = th0[:3]
            th2_0 = th0[3:6]
            th3_0 = th0[6:9]
            th4_0 = th0[9:]
            th0 = [th1_0, th2_0, th3_0, th4_0]
            print(f"TH0: {th0}")
            folder_name = 'theta'
            os.makedirs(folder_name, exist_ok=True)
            theta_config = folder_name + "/theta_0.json"
            with open(theta_config, "w") as outfile:
                outfile.write(th0)
            # grab cable lengths of all 9 motors
            l = grab_arm_cable_lens(Arm.get_position(), l, l0, th0, r
               )
            print(f"Cable lengths based off of dtheta: {l}\n")
            q = grab_arm_q(l[0], l[1], l[2], l[3], BigMotors[0],
                           BigMotors[1], BigMotors[2], BigMotors[3],
                              s, d)
            # Report our initial configuration
            print(f"Our current q: {q}\n")
        # print out the length changes
        elif key_input == chr(WKEY_ASCII_VALUE) or key_input == chr(
           TKEY_ASCII_VALUE):
            pos = Arm.get_position()
            l = grab_arm_cable_lens(pos, l, l0, th0, r)
            joint_th = Joints.get_position()
            mj0 = joint_th[0] - base_offset
            mj1 = joint_th[1] - offset1
            mj2 = joint_th[2] - offset2
            mj3 = joint_th[3] - offset3
            q = grab_arm_q(l[0], l[1], l[2], l[3], mj0, mj1, mj2, mj3
               , s, d)
            q_data = np.repeat(q, 10, axis=1)
```

```
            print(f"Q DATA SIZE: {q_data.shape}")
            tau_data = np.zeros((nq, 1))
            timestamps = np.zeros((1, 1))
            c_data = np.zeros((nmod, 1))
            # Open up controller parameters
            param, config_params = parse_config()
            # Open up desired q from json file
            qd_str, qd_params = parse_setpoint()
            qd = grab_qd(qd_str)
            qd_mat = mat2np('tube_up/qd.mat', 'qd')
            dqd_mat = mat2np('tube_up/dqd.mat', 'dqd')
            ddqd_mat = mat2np('tube_up/ddqd.mat', 'ddqd')
            tvec = mat2np('tube_up/tvec.mat', 'tvec')
            zero = np.zeros((16, 1))
            t_old = time.time()
            # our loop's "starting" time
            t_0 = time.time()
            while 1:
                if kbhit():
                    c = getch()
                    first_time = True
                    for i in range(10):
                        Arm.send_torque_cmd(nmod * [0])
                        Joints.send_torque_cmd(nm * [0])
                        print("[Q KEY PRESSED] : All motors stopped\n
                            ")
                    print(f"Q DATA SIZE: {q_data.shape}")
                    save_data(q_data, qd, tau_data, c_data, t_0,
                              timestamps, config_params, qd_params,
                                 dt_loop)
                    break
                else:
                    # grab current time
                    pos = Arm.get_position()
                    l = grab_arm_cable_lens(pos, l, l0, th0, r)
                    joint_th = Joints.get_position()
                    mj0 = joint_th[0] - base_offset
                    mj1 = joint_th[1] - offset1
                    mj2 = joint_th[2] - offset2
                    mj3 = joint_th[3] - offset3
                    q = grab_arm_q(l[0], l[1], l[2], l[3],
                                   mj0, mj1, mj2, mj3, s, d)
                    q_data = np.append(q_data, q, axis=1)
                    if key_input == chr(TKEY_ASCII_VALUE):
                        n = np.argmax(tvec > time.time() - t_0) - 1
                        qd = np.array(qd_mat[:, n]).reshape(-1, 1)
                        dqd = np.array(dqd_mat[:, n]).reshape(-1, 1)
```

```
                        ddqd = np.array(ddqd_mat[:, n]).reshape(-1,
                            1)
                    else:
                        dqd = zero
                        ddqd = zero
                    if first_time:
                        dq = np.zeros((nq, 1))
                        first_time = False
                    else:
                        t = time.time()
                        time_elapsed = t-t_0
                        timestamps = np.append(timestamps,
                            time_elapsed)
                        dt = t - t_old
                        print(f"[DEBUG] dt: {dt}\n")
                        t_old = t
                        dq = diff(q, q_old, dt)
                    q_old = q
                    # calculate errors
                    err = q - qd
                    err_dot = dq
                    # tau_test, cont = puppet_controller(q,dq,qd,zero
                        ,zero,d,hp,mplate,r,s,param,Lm=Lm)
                    tau, cont = puppet_controller_wrapper(
                        q, dq, qd, dqd, ddqd, d, hp, mplate, r, s,
                            param, puppet_controller_c, Lm=Lm)
                    c_data = np.append(c_data, cont, axis=1)
                    tau_data = np.append(tau_data, tau, axis=1)
                    arm_input, mod_cmds = torque_to_current(tau, 1)
                    Arm.send_torque_cmd(mod_cmds)
                    joint_cmds = [arm_input[0], -
                                  arm_input[4], -arm_input[8], -
                                     arm_input[12]]
                    Joints.send_torque_cmd(joint_cmds)
        elif key_input == chr(NKEY_ASCII_VALUE):
            # Update to new config
            with open('config.json') as config:
                param = json.load(config)
            print(param)
            k1 = param['k1']
            k2 = param['k2']
            k3 = param['k3']
            phi1 = param['phi1']
            phi2 = param['phi2']
            phi3 = param['phi3']
            dL1 = param['dL1']
            dL2 = param['dL2']
```

```
            dL3 = param['dL3']
            jm0 = param['jm0']
            jm1 = param['jm1']
            jm2 = param['jm2']

    print("[END OF PROGRAM] Disabling torque\n")
    # Disable Dynamixel Torque
    Arm.disable_torque()
    Joints.disable_torque()
    # Close port
    portHandlerMod.closePort()
except Exception:
    print("[ERROR] Disabling torque\n")
    Arm.disable_torque()
    Joints.disable_torque()
    traceback.print_exc()
```

Bibliography

[1] D. Rus and M. T. Tolley, "Design, fabrication and control of soft robots," *Nature*, vol. 521, no. 7553, pp. 467–475, May 2015, ISSN: 1476-4687. DOI: 10.1038/nature14543. URL: https://doi.org/10.1038/nature14543.

[2] U. Culha, J. Hughes, A. Rosendo, F. Giardina, and F. Iida, "Design Principles for Soft-Rigid Hybrid Manipulators," en, in *Soft Robotics: Trends, Applications and Challenges*, C. Laschi, J. Rossiter, F. Iida, M. Cianchetti, and L. Margheri, Eds., ser. Biosystems & Biorobotics, Cham: Springer International Publishing, 2017, pp. 87–94, ISBN: 978-3-319-46460-2. DOI: 10.1007/978-3-319-46460-2_11.

[3] J. M. Bern, L. Z. Yañez, E. Sologuren, and D. Rus, "Contact-rich soft-rigid robots inspired by push puppets," in *2022 IEEE 5th International Conference on Soft Robotics (RoboSoft)*, 2022, pp. 607–613. DOI: 10.1109/RoboSoft54090.2022.9762203.

[4] J. Wyneken, *The Anatomy of Sea Turtles The Anatomy of Sea Turtles*. Dec. 2001.

[5] C. Della Santina, C. Duriez, and D. Rus, "Model-based control of soft robots: A survey of the state of the art and open challenges," *IEEE Control Systems*, vol. 43, no. 3, pp. 30–65, Jun. 2023, ISSN: 1941-000X. DOI: 10.1109/mcs.2023.3253419. URL: http://dx.doi.org/10.1109/MCS.2023.3253419.

[6] T. Du, J. Hughes, S. Wah, W. Matusik, and D. Rus, "Underwater soft robot modeling and control with differentiable simulation," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4994–5001, 2021. DOI: 10.1109/LRA.2021.3070305.

[7] J. M. Bern, F. Zargarbashi, A. Zhang, J. Hughes, and D. Rus, "Simulation and fabrication of soft robots with embedded skeletons," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 5205–5211. DOI: 10.1109/ICRA46639.2022.9811844.

[8] Y. Sun, Y. Jiang, H. Yang, L.-C. Walter, J. Santoso, E. H. Skorina, and C. Onal, "Salamanderbot: A soft-rigid composite continuum mobile robot to traverse complex environments," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 2953–2959. DOI: 10.1109/ICRA40945.2020.9196790.

[9] W. R. Wockenfuß, V. Brandt, L. Weisheit, and W.-G. Drossel, "Design, modeling and validation of a tendon-driven soft continuum robot for planar motion based on variable stiffness structures," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 3985–3991, 2022. DOI: 10.1109/LRA.2022.3149031.

[10] Y. Yang, Y. Li, and Y. Chen, "Principles and methods for stiffness modulation in soft robot design and development," en, *Bio-Design and Manufacturing*, vol. 1, no. 1, pp. 14–25, Mar. 2018, ISSN: 2522-8552. DOI: 10.1007/s42242-018-0001-6. URL: https://doi.org/10.1007/s42242-018-0001-6 (visited on 05/08/2023).

[11] D. Bruder, M. A. Graule, C. B. Teeple, and R. J. Wood, "Increasing the payload capacity of soft robot arms by localized stiffening," *Science Robotics*, vol. 8, no. 81, eadf9001, 2023. DOI: 10.1126/scirobotics.adf9001. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.adf9001. URL: https://www.science.org/doi/abs/10.1126/scirobotics.adf9001.

[12] A. Stilli, H. A. Wurdemann, and K. Althoefer, "Shrinkable, stiffness-controllable soft manipulator based on a bio-inspired antagonistic actuation principle," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2476–2481. DOI: 10.1109/IROS.2014.6942899.

[13] W. Zhu, C. Lu, Q. Zheng, Z. Fang, H. Che, K. Tang, M. Zhu, S. Liu, and Z. Wang, "A soft-rigid hybrid gripper with lateral compliance and dexterous in-hand manipulation," *CoRR*, vol. abs/2110.10035, 2021. arXiv: 2110.10035. URL: https://arxiv.org/abs/2110.10035.

[14] F.-Y. Xu, F.-Y. Jiang, Q.-S. Jiang, and Y.-X. Lu, "Soft actuator model for a soft robot with variable stiffness by coupling pneumatic structure and jamming mechanism," *IEEE Access*, vol. 8, pp. 26 356–26 371, 2020. DOI: 10.1109/ACCESS.2020.2968928.

[15] M. A. Robertson and J. Paik, "New soft robots really suck: Vacuum-powered systems empower diverse capabilities," *Science Robotics*, vol. 2, no. 9, eaan6357, 2017. DOI: 10.1126/scirobotics.aan6357. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aan6357. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aan6357.

[16] Y. Jung, K. Kwon, J. Lee, and S. H. Ko, "Untethered soft actuators for soft standalone robotics," *Nature Communications*, vol. 15, no. 1, p. 3510, Apr. 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-47639-0. URL: https://doi.org/10.1038/s41467-024-47639-0.

[17] Y. Jung, K. Kwon, J. Lee, and S. H. Ko, "Untethered soft actuators for soft standalone robotics," *Nature Communications*, vol. 15, no. 1, p. 3510, Apr. 2024, ISSN: 2041-1723. DOI: 10.1038/s41467-024-47639-0. URL: https://doi.org/10.1038/s41467-024-47639-0.

[18] R. M. Murray, Z. Li, S. S. Sastry, and S. S. Sastry, *A mathematical introduction to robotic manipulation*. CRC press, 1994.

[19] C. Della Santina, C. Duriez, and D. Rus, "Model Based Control of Soft Robots: A Survey of the State of the Art and Open Challenges," en, Oct. 2021. URL: https://arxiv-org.cmu.idm.oclc.org/abs/2110.01358v1 (visited on 10/09/2021).

[20] C. Della Santina, R. K. Katzschmann, A. Bicchi, and D. Rus, "Model-based dynamic feedback control of a planar soft robot: Trajectory tracking and interaction with the environment," *The International Journal of Robotics Research*, vol. 39, no. 4, pp. 490–513, 2020, Publisher: SAGE Publications Sage UK: London, England.

[21] S. M. H. Sadati, S. E. Naghibi, I. D. Walker, K. Althoefer, and T. Nanayakkara, "Control space reduction and real-time accurate modeling of continuum manipulators using ritz and ritz–galerkin methods," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 328–335, 2018. DOI: 10.1109/LRA.2017.2743100.

[22] V. Falkenhahn, A. Hildebrandt, R. Neumann, and O. Sawodny, "Model-based feedforward position control of constant curvature continuum robots using feedback linearization," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 762–767. DOI: 10.1109/ICRA.2015.7139264.

[23] F. Renda, F. Boyer, J. Dias, and L. Seneviratne, "Discrete cosserat approach for multisection soft manipulator dynamics," *IEEE Transactions on Robotics*, vol. 34, no. 6, pp. 1518–1533, 2018. DOI: 10.1109/TRO.2018.2868815.

[24] J. M. Bern and D. Rus, "Soft ik with stiffness control," in *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*, 2021, pp. 465–471. DOI: 10.1109/RoboSoft51838.2021.9479195.

[25] P. Polygerinos, Z. Wang, J. T. B. Overvelde, K. C. Galloway, R. J. Wood, K. Bertoldi, and C. J. Walsh, "Modeling of soft fiber-reinforced bending actuators," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 778–789, 2015. DOI: 10.1109/TRO.2015.2428504.

[26] J. M. Bern, P. Banzet, R. Poranne, and S. Coros, "Trajectory optimization for cable-driven soft robot locomotion.," in *Robotics: Science and Systems*, vol. 1, 2019.

[27] Z. J. Patterson, C. D. Santina, and D. Rus, *Modeling and control of intrinsically elasticity coupled soft-rigid robots*, 2023. eprint: arXiv:2311.05362.

[28] H. Zhang, R. Cao, S. Zilberstein, F. Wu, and X. Chen, "Toward effective soft robot control via reinforcement learning," in *Intelligent Robotics and Applications*, Y. Huang, H. Wu, H. Liu, and Z. Yin, Eds., Cham: Springer International Publishing, 2017, pp. 173–184, ISBN: 978-3-319-65289-4.

[29] C. Della Santina, C. Duriez, and D. Rus, "Model-based control of soft robots: A survey of the state of the art and open challenges," *IEEE Control Systems Magazine*, vol. 43, no. 3, pp. 30–65, 2023. DOI: 10.1109/MCS.2023.3253419.

[30] R. K. Katzschmann, J. DelPreto, R. MacCurdy, and D. Rus, "Exploration of underwater life with an acoustically controlled soft robotic fish," *Science Robotics*, vol. 3, no. 16, eaar3449, 2018. DOI: 10.1126/scirobotics.aar3449. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.aar3449. URL: https://www.science.org/doi/abs/10.1126/scirobotics.aar3449.

[31] N. van der Geest, L. Garcia, F. Borret, R. Nates, and A. Gonzalez, "Soft-robotic green sea turtle (chelonia mydas) developed to replace animal experimentation provides new insight into their propulsive strategies," *Scientific Reports*, vol. 13, no. 1, p. 11 983, Jul. 2023, ISSN: 2045-2322. DOI: 10.1038/s41598-023-37904-5. URL: https://doi.org/10.1038/s41598-023-37904-5.

[32] N. van der Geest, L. Garcia, R. Nates, and D. A. Godoy, "New insight into the swimming kinematics of wild green sea turtles (chelonia mydas)," en, *Sci. Rep.*, vol. 12, no. 1, p. 18 151, Oct. 2022.

[33] J. S. Cely, R. Saltaren, G. Portilla, O. Yakrangi, and A. Rodriguez-Barroso, "Experimental and computational methodology for the determination of hydrodynamic coefficients based on free decay test: Application to conception and control of underwater robots," *Sensors*, vol. 19, no. 17, 2019, ISSN: 1424-8220. DOI: 10.3390/s19173631. URL: https://www.mdpi.com/1424-8220/19/17/3631.

[34] A. Crespi, D. Lachat, A. Pasquier, and A. Ijspeert, "Controlling swimming and crawling in a fish robot using a central pattern generator," *Autonomous Robots*, vol. 25, Aug. 2008. DOI: 10.1007/s10514-007-9071-6.

[35] H. Deng, P. Burke, D. Li, and B. Cheng, "Design and experimental learning of swimming gaits for a magnetic, modular, undulatory robot," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2021, pp. 9562–9568. DOI: 10.1109/IROS51168.2021.9636100.

[36] A. I. Zharinov, Y. A. Tsybina, and S. Y. Gordleeva, *Review: Cpg as a controller for biomimetic floating robots*, 2021. arXiv: 2112.07295 [q-bio.NC].

[37] Z. J. Patterson, W. Xiao, E. Sologuren, and D. Rus, *Safe control for soft-rigid robots with self-contact using control barrier functions*, 2024. arXiv: 2311.03189 [cs.RO].

[38] J. Wang and A. Chortos, "Control strategies for soft robot systems," *Advanced Intelligent Systems*, vol. 4, no. 5, p. 2 100 165, 2022. DOI: https://doi.org/10.1002/aisy.202100165. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/aisy.202100165. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/aisy.202100165.

[39] B. Jones and I. Walker, "Kinematics for multisection continuum robots," *IEEE Transactions on Robotics*, vol. 22, no. 1, pp. 43–55, 2006. DOI: 10.1109/TRO.2005.861458.

[40] C. Della Santina, A. Bicchi, and D. Rus, "On an improved state parametrization for soft robots with piecewise constant curvature and its use in model based control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1001–1008, 2020. DOI: 10.1109/LRA.2020.2967269.

[41] R. Featherstone, *Rigid body dynamics algorithms*. Springer, 2014.

[42] V. Santibañez and R. Kelly, "Global asymptotic stability of the PD control with computed feedforward in closed loop with robot manipulators," *IFAC Proceedings Volumes*, 14th IFAC World Congress 1999, Beijing, Chia, 5-9 July, vol. 32, no. 2, pp. 683–688, Jul. 1999, ISSN: 1474-6670. DOI: 10.1016/S1474-6670(17)56116-9. (visited on 02/02/2024).

[43] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, "Benchmarking in manipulation research: Using the yale-cmu-berkeley object and model set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[44] L. A. Hawkes, O. Exeter, S. M. Henderson, C. Kerry, A. Kukulya, J. Rudd, S. Whelan, N. Yoder, and M. J. Witt, "Autonomous underwater videography and tracking of basking sharks," *Animal Biotelemetry*, vol. 8, no. 1, p. 29, Aug. 2020, ISSN: 2050-3385. DOI: 10.1186/s40317-020-00216-w. URL: https://doi.org/10.1186/s40317-020-00216-w.

[45] N. Konow, J. A. Cheney, T. J. Roberts, J. R. S. Waldman, and S. M. Swartz, "Spring or string: Does tendon elastic action influence wing muscle mechanics in bat flight?" en, *Proc. Biol. Sci.*, vol. 282, no. 1816, p. 20 151 832, Oct. 2015.

[46] J. Collins, S. Chand, A. Vanderkop, and D. Howard, "A review of physics simulators for robotic applications," *IEEE Access*, vol. 9, pp. 51 416–51 431, 2021. DOI: 10.1109/ACCESS.2021.3068769.

[47] G. Bellegarda and A. Ijspeert, *Cpg-rl: Learning central pattern generators for quadruped locomotion*, 2022. arXiv: 2211.00458 [cs.RO].

[48] A. Raffin, O. Sigaud, J. Kober, A. Albu-Schäffer, J. Silvério, and F. Stulp, *An open-loop baseline for reinforcement learning locomotion tasks*, 2024. arXiv: 2310.05808 [cs.RO].

[49] G. Li, A. Ijspeert, and M. Hayashibe, "Ai-cpg: Adaptive imitated central pattern generators for bipedal locomotion learned through reinforced reflex neural networks," *IEEE Robotics and Automation Letters*, vol. 9, no. 6, pp. 5190–5197, 2024. DOI: 10.1109/LRA.2024.3388842.

[50] S. M. Youssef, M. Soliman, M. A. Saleh, A. H. Elsayed, and A. G. Radwan, "Design and control of soft biomimetic pangasius fish robot using fin ray effect and reinforcement learning," *Scientific Reports*, vol. 12, no. 1, p. 21 861, Dec. 2022, ISSN: 2045-2322. DOI: 10.1038/s41598-022-26179-x. URL: https://doi.org/10.1038/s41598-022-26179-x.

[51] J. Wang, E. Uchibe, and K. Doya, "Em-based policy hyper parameter exploration: Application to standing and balancing of a two-wheeled smartphone robot," *Artificial Life and Robotics*, vol. 21, no. 1, pp. 125–131, Mar. 2016, ISSN: 1614-7456. DOI: 10.1007/s10015-015-0260-7. URL: https://doi.org/10.1007/s10015-015-0260-7.

[52] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, J. Peters, and J. Schmidhuber, "Policy gradients with parameter-based exploration for control," in *Artificial Neural Networks - ICANN 2008*, V. Kůrková, R. Neruda, and J. Koutník, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 387–396, ISBN: 978-3-540-87536-9.

[53] J. Peters and S. Schaal, "Reinforcement learning by reward-weighted regression for operational space control," in *International Conference on Machine Learning*, 2007. URL: https://api.semanticscholar.org/CorpusID:11551208.

[54] B. E. Jackson, K. Tracy, and Z. Manchester, "Planning with attitude," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5658–5664, 2021. DOI: 10.1109/LRA.2021.3052431.

[55] F. Stella, J. Hughes, D. Rus, and C. Della Santina, "Prescribing cartesian stiffness of soft robots by co-optimization of shape and segment-level stiffness," *Soft Robotics*, vol. 10, no. 4, pp. 701–712, 2023, PMID: 37130308. DOI: 10.1089/soro.2022.0025. eprint: https://doi.org/10.1089/soro.2022.0025. URL: https://doi.org/10.1089/soro.2022.0025.