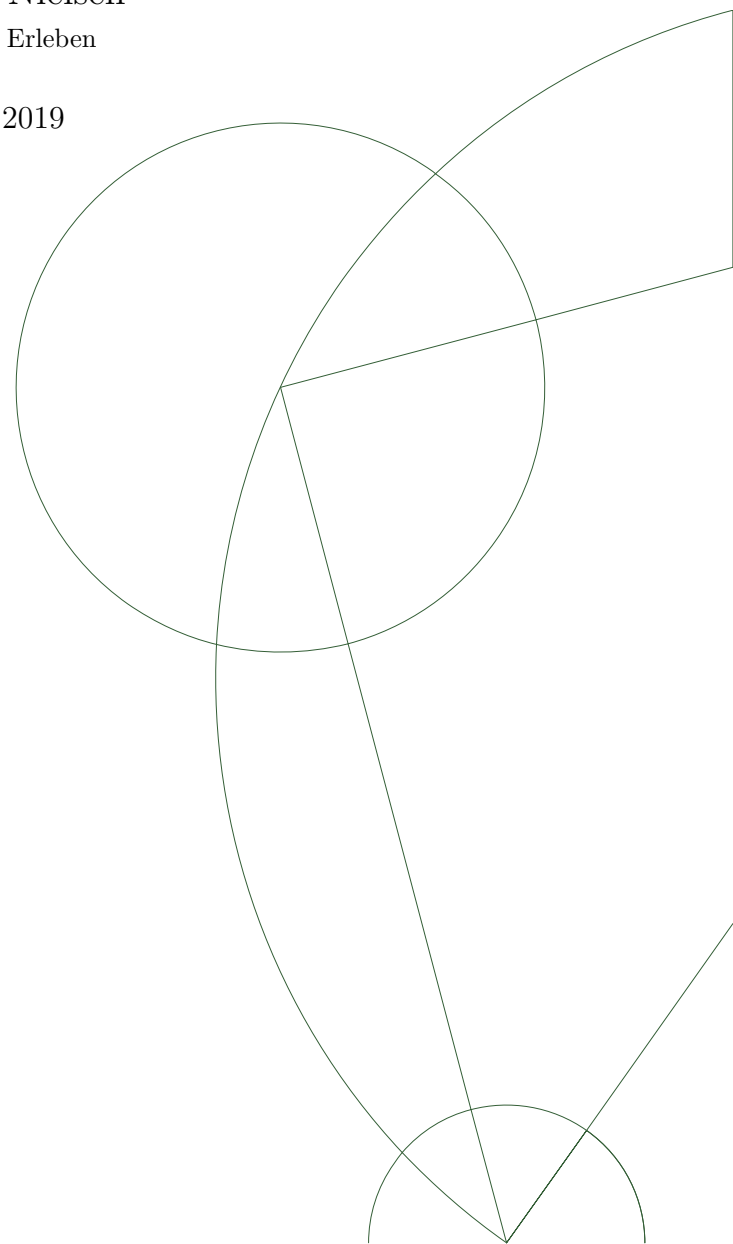
UNIVERSITY OF COPENHAGEN

Automation of Control and Design Optimisation for Soft Robots manufactured with Strong Feasibility Guarantees

Max Kragballe Nielsen
Supervisor: Kenny Erleben

September 1, 2019



Abstract

In this work, we present models for simulating cable and pressure forces, which allows for the interactive exploration of the action space of soft robots. Furthermore, we formulate two strategies for learning to solve control tasks in a simulated environment, and investigate how well the learned policies transfer from simulations to reality using the Learning Cube framework.

We have implemented our actuation models in a position based dynamics simulator, and fabricated physical copies of the digital models used in the simulations. Our results indicate that the cable model can be used to gain a good approximation of how the robot will behave in the real world, and transferring control policies to a physical copy of the robot proved feasible. The pressure model were found to be unstable, which limited its usability. However, we were still able to gain a limited understanding of pressure driven robot motion prior to fabrication, but producing more robust pressure simulations needs to be investigated further in future work.

The take-away of this thesis, is a possible improvement of the design process of soft robots, since we are able to quickly prototype robot designs that we can learn how to control in a simulated environment, and that we can strongly guarantee can be fabricated.

Contents

Chapter 1

Introduction

Industrial robots are playing a vital part in modern society, due to their ability to perform repetitive tasks faster, cheaper and more accurately than human workers. Traditionally robots have been confined to work in factories and laboratories, but with the advent of collaborative robots [PC99], we are beginning to see robots becoming an ever-increasing part of our daily routines. Even though collaborative robots are more affordable than traditional industrial robots [AS], they are still not commonly available to the average consumer. However, with the coming of increasingly sophisticated consumer-level 3D printers [Ind; BVb; PRI18], personal manufacturing of sophisticated *soft bodied* robots is now possible. As their name implies, *soft robots* consist of an outer frame or shell made of a soft material such as silicone elastomer. The soft robots are actuated to create desirable deformations in the soft body, which makes them well-suited for completing tasks with high variability such as grasping irregular bodies [Gli+18], interacting with small objects [SIT91] or sampling corals from deep reefs [Gal+16].

Learning to control soft robots efficiently is a difficult task, due to their flexibility and adaptability, with the choice of control method often being determined based on the design of the robot. Machine learning techniques have been used to learn inverse kinematics of robots, as in [Rie+09; Hol+18]. In [Hol+18] a platform for efficiently collecting data about robot motion was introduced, as well as a method that was robust towards shortcomings in the robot manufacturing process. However, these shortcomings can be avoided by increasing the quality of the manufacturing process, which enables the usage of 'digital twins'. 'Digital twins' have been used in previous works to simulate the behaviour of physical robots [Dur13; Hao+17] to great success. Simulations increase the efficiency of the prototyping pipeline, as researchers are now beginning to be able to explore the configuration space of their robots before manufacturing a physical copy. This is nonetheless only an advantage as long as the behaviour of the simulated models match their physical world counterparts. Observe in Figure 1.1 some examples of digital models and their physical silicone counter-parts.

Figure 1.1: Soft robot digital models and their physical counter-parts.

Simulation is an important tool in a huge amount of applications, and the goal of a simulation is essentially to build a digital world identical to reality. However, the amount of variable factors in the real world makes perfect simulations impossible, which means that we can only simulate approximations of real-life behaviour. The inaccuracies are caused by a number of different factors, but are mainly governed by the fact that we are using Newtonian laws - and thus, disregarding quantum laws and relativity - and that we are approximating temporal derivatives using tiny, but not infinitesimal, time steps. This creates a gap between simulations and reality, which needs to be closed if we want to transfer what we learn in the simulated environement to the real world.

1.1 Previous Work in Robot Control and Design

Learning to control robots and learning optimal control policies have been examined in works as early as [Mat97], in which the robots were rigid and contained a noisy and imperfect internal representation of their environments. The challenge was to overcome the noise of the simulated environment to complete their tasks in the real world. In more recent work, training policies on a large diversity of simulated scenarios by randomisation of task parameters and performing real world roll-outs, has shown considerable promise in terms of transfer of skills to the real world [Che+19; Ope+18; CVM18]. However, similarly for both most old and recent work is that researchers make use of rigid high-precision robots with a well-known configuration space.

For soft robots, the action space is defined by the geometry of the soft body and the placement of actuators, which means that learning to control the robot optimally might not be sufficient in terms of completing a task. Instead, the design and control need to be simultaneously optimised, resulting in the dual problem that is the design of robots capable of solving specific tasks.

In recent years, research in reinforcement learning has increased the efficiency of control policy learning substantially. Computers are now able to learn tasks previously thought only to be possible by humans, such as defeating world champions of esports games [Pac+], creating art [HHZ19] and turning simple drawings into photo-realistic sceneries [Par+19]. Already in [HL12] researchers began to explore the usage of reinforcement learning algorithms in terms of automating the design of soft robots. In [Che+14] this concept was further investigated and promising results in terms of simulated design optimation using evolutionary strategies were showcased. The caveat of this work was that the robots that evolved were almost impossible to manufacture, even though more recent work [Kri+19] has shown progress towards manufacturing the evolved designs.

As previously mentioned, we often observe several disparities between simulations and reality - the so-called 'simulation-to-reality gap' - which hinders direct transfer of control policies to the real world. In recent years, a lot of research has been devoted to bridge this gap, including learning from real world observations [Lev+16], imitating expert human examples [Zho+19], real-world roll-outs [Che+19], and zero-shot transfer approaches using domain randomisation [JDJ17] have all shown impressive results.

The main goal of this thesis is to explore the possibilities of automating the control and design of soft robots. Due to the sequential nature of physical systems, we want to perform all training inside a simulated environment, since this will allow us to increase the amount of training we can perform, as well as allowing us to explore different designs prior to fabrication. However, to achieve this goal, we need to solve the following problems:

- When we design a digital model of a robot, we need to be able to tell if it is feasible to

fabricate the robot.

- We need to be able to capture robot motion data from the real world in order to validate our simulation results.
- To learn to control the soft robots, we need to be able to simulate the physical actuators working on the soft bodies.

Once these problems have been solved, we can then train a control or design model, manufacture a physical copy, and verify whether the control policy transfers to the real world or not. However, as we recognise that bridging the simulation-to-reality gap might be too ambitious for this thesis, we will merely investigate the gap, focusing on the reasons as to why it appears.

1.2 Thesis Structure

In Chapter 2, we will begin by examining some of the previous work on the design and manufacturing of soft robots, and describe in detail a geometric constraint model for determining the robustness of a design. We will then describe the Learning Cube environment in Chapter 3, which is the platform we will be using for collecting data of the motion of our manufactured models. We move on to derive models for simulating cable and pressure forces in Chapters 4 and 5, respectively, and in Chapter 6 we describe how the action space of a soft robot can be explored interactively, and examine the reality gap in our simulations. Here, we will also discuss the limitations of our robust manufacturing model and the actuation models. Finally, we will develop a model for learning control policies in Chapter 7 and examine how the policies we learn, transfer to the real world.

1.3 Preface

This thesis is written to a master student level computer scientist with limited understanding of simulations and computational geometry, and a strong foundation in reinforcement learning, numerical optimisation, and general machine learning principles. Due to the usage of NVIDIA Flex [Cor19b], the simulation software cannot be made available, but implementations of the actuation and control models described in Chapter 4, 5 and 7 can be obtained by contacting either author or supervisor. Videos of all simulations from the Flex simulator presented in this thesis are available at `mystiking.github.io`, where videos of the physical experiments are also present.

Chapter 2

Designing and Manufacturing Soft Robots

In the field of soft robotics many of the limitations that researchers currently face are dictated by the tedious design and manufacturing processes needed to fabricate usable robots. Tools for designing soft robots are beginning to surface, with the development of tools such as VoxCAD [Che+14] - where it is possible to use non-homogeneous materials - but researchers are currently mainly using conventional computer-aided design (CAD) tools for modelling their soft robots. These tools include well-known conventional 3D modelling software such as Blender [Ble] and Maya [Aut], but also tools relying more heavily on geometric descriptions, such as OpenSCAD [KW10].

In [Che+14] researchers began automating the design process by using evolutionary strategies to 'evolve' a good design instead of creating it manually. The soft robots that evolved scored high on their fitness measure in terms of completing a specified task - locomotion - but it was not feasible to actually manufacture the soft robots. The article built on top of the work of [HL12], where a similar - but simpler - strategy was used to generate soft robots that were actually manufacturable. The robots could only be actuated inside a pressure chamber though. To create a soft robot of heterogeneous materials additive manufacturing is necessary, but at this point multi-material 3D printers, such as the StrataSys J750 [Ltd], are too expensive to be available for the average consumer, with prices starting at $\sim 250,000$ USD [Ltd19]. Soft robots are actuated to create desirable deformations in their soft bodies, and the usage of heterogeneous materials have proved to work well in terms of controlling these deformations [Gal+16; Gli+18]. However, with the increasing availability of consumer-level 3D-printers, creating molds for casting soft bodies has quickly become one of the favoured methods for fast prototyping of soft robot designs [RT15; Lee+17]. Heterogeneous materials are then incorporated into the soft bodies of the robots by diving the fabrication process into multiple castings, and play a vital role in terms of controlling how the soft robot deform during actuation. This has led to a lot of soft robots being designed as layered constructs [RT15].

Robot designs have also been presented where other materials than a silicone-like elastomer was used to create the soft shell of the robot. In [BKC17] robots were created using fabric with polyester filling - in other words, plushies - which were shown to work well with cable driven actuation. These 'plush robot' are not limited to be layered constructs in the same way as silicone casts, but manufacturing a plush robot requires a lot of practical skills - i.e. experience in needlework - which makes prototyping slow. In [AKY18] inflated balloon-like robots were created, that showed

promise in terms of especially human-robot interaction, as the soft material of the outer shell creates a safe environment for the human interacting with the robot. However, the manufacturing process is difficult to inexperienced users, which again limits the potential of the robots.

The process of producing a soft robot requires a large amount of design-knowledge, but in recent years the usability of 3D printers have increased manifold. This means that operating 3D printers have become a lot easier, but novice manufacturers still face the problem of designing molds from which a soft robot can be removed without damaging its soft body. In this chapter we will formulate a model that can give a strong guarantee on the feasibility of a robot design, enabling us to discard invalid designs without having to fabricate any physical copies.

2.1 Conservative Safe Manufacturing

For this thesis we will be using the design model we created during a pre-project, in order to construct molds for casting soft silicone robots. Therefore, this section will act as a 'recap' of the contents of that project. The tool contains two different implementations of constructive solid geometry (CSG) - one using libigl [Jac+18] to represent solids as surface meshes, and one using signed distance maps. For the surface mesh implementation, the end product is the surface mesh representing the root of the tree of the boolean operators defining the model. Observe in Figure 2.1 an example of a tree of boolean operators.

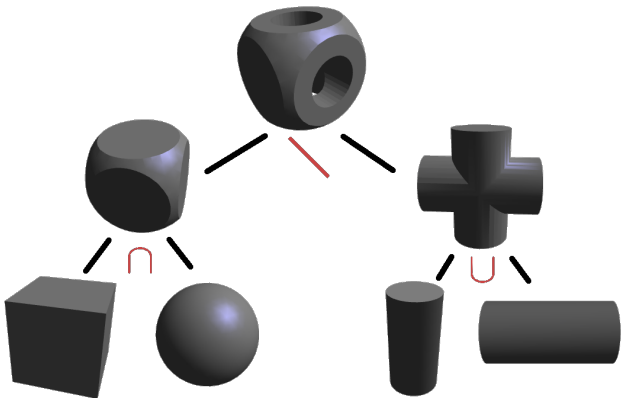


Figure 2.1: The boolean CSG operators. The objects were created using the tool created during the pre-project.

For the signed distance map implementation, the end result is the distance function corresponding to the compound of all the CSG operations. As we don't have to evaluate this function during compounding, it is very fast to compute, but as we then need to triangulate the distance map to create a surface mesh, the two implementations are roughly equal in terms of speed for medium sized robots.

To construct the mold for casting the soft robots we can either model the mold directly in CSG or use the difference operator to create a suitable 'hole' corresponding to the negative of the soft body in a solid block. In either case, we want to make sure that the robot can be removed from the mold without the soft body suffering any damages, as the production of the mold and the casting of the silicone is a time-consuming and tedious process. To ensure that the risk of a robot rupturing is minimised, we propose a geometric constraint model that can provide a strong

feasibility guarantee by numerically examining a set of empirically derived *robustness conditions*. The robustness conditions from the pre-project were:

- A solution to the mold extraction problem must exist for the geometry of the robot.
- The internal distances - i.e. the *thickness* - of the robot must not be too small or else a too weak structure will develop, and the robot is at great risk of being damaged during extraction.
- The sharpness of angles between convex surface parts must not be too extreme or else fracture tips will be present in the robot.
- The cable shielding inside a robot must not bend beyond a maximum curvature limit. Otherwise the cable shielding will begin to buckle, which means that actuation devices cannot be added to the robot, or at least will not function at optimal capacity.

When these conditions are satisfied, a robot design is deemed *feasible*. This does not mean that a design that satisfies all conditions are guaranteed to be manufacturable, but instead that there is a high probability that the fabrication process will succeed. The first of the robustness conditions can be seen as a necessary condition where as the remaining ones are somewhat relaxed necessary conditions based on empirical evidence.

2.1.1 Mold Extraction Feasibility

The first condition of the geometric constraint model is that a solution to the mold extraction problem must exist [Ber+08]. Given a mold with a cavity corresponding to the 3D polyhedron $\mathcal{P}$ of the robot shape, we must be able to remove the robot by performing a single translation without intersecting any of the walls of the mold. For our molds the polygon $\mathcal{P}$ must always have a top facet, f_0, parallel with the xy-plane with corresponding normal $\mathbf{n}_0$. If this was not the case, the silicone would not be able to stay within the mold, and therefore, it is a necessary condition that f_0 must exist. Therefore, an extraction direction, $\mathbf{d}$, is valid if and only if it makes a 90° angle with all normals of the ordinary facets of the polyhedron, $\mathcal{P}$. When we solve the mold extraction problem, we therefore need to optimise for the extraction direction, $\mathbf{d}$, such that,

$$\mathbf{d}^* \equiv \arg\min_{\mathbf{d}} \sum_{\mathbf{n}\in\mathcal{P}\setminus\{\mathbf{n}_0\}} \mathbf{d}_x\mathbf{n}_x + \mathbf{d}_y\mathbf{n}_y + \mathbf{n}_z \tag{2.1}$$

subject to

$$\mathbf{d}_x\mathbf{n}_x + \mathbf{d}_y\mathbf{n}_y + \mathbf{n}_z \leq 0,\ \forall \mathbf{n} \in \mathcal{P}, \mathbf{n} \neq \mathbf{n}_0 \tag{2.2}$$

in which case the solution will be the 'safest' translation direction in terms of avoiding intersecting the sides of the mold.

We are only interested in testing the feasibility conditions as we let the user specify an extraction direction which we test using Equation (2.2). The mould extraction problem is formulated for non-elastic materials, which means that the lack of a solution to Equation (2.2) does not guarantee that the robot cannot be removed from its mould, due to the elasticity of the silicone. However, when a solution exists a strong guarantee can be given on the feasibility of the removal process.

2.1.2 Thickness Constraints

Another problem that can affect the feasibility of removing a robot from its mould is when the soft body contains too fine internal structures. In other words, when there exists low mass areas of silicone, the soft body of the robot might rupture during removal. Observe in Figure 2.2 an example of a ruptured robot, caused by too thin structures.

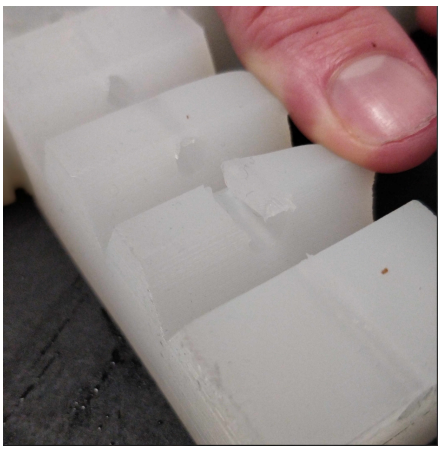


Figure 2.2: Robot that ruptured due to too thin structures.

To avoid this kind of rupturing, the second robustness condition is introduced. This condition requires that a soft robot must be sufficiently 'thick' everywhere in order to strongly guarantee that the robot can be manufactured.

From morphology [Jäh05] we know that the medial axis representation - or topological skeleton - of a polyhedron can be used to measure the thickness of an object, since the medial axis is the set of all points having more than one closest point on the surface of the robot. Therefore, the internal distances can be verified to be safe by measuring the distance to the surface from every point on the medial axis. Observe in Figure 2.3 the medial axis representation (in 2D) of a shape, which can be used to measure thickness.

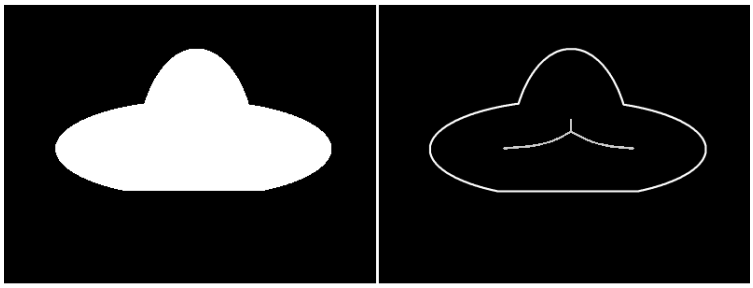


Figure 2.3: The medial axis representation (**right**) of the shape on the **left**. The medial axis is depicted as the skeleton-like white lines within the outlined shape.

More formally, given the medial axis $\mathcal{M}$ and the limit $\delta_{internal}$, a robot design must satisfy

$$\delta_{surface}(\mathbf{p}) \geq \frac{\delta_{internal}}{2}, \quad \forall \mathbf{p} \in \mathcal{M} \tag{2.3}$$

where $\delta_{surface}(\mathbf{p})$ is the smallest distance from the point $\mathbf{p}$ to the surface of the robot.

This approach works well for polyhedrons without sharp corners, but when the robot contains sharp features the medial axis can no longer be used to determine if the internal distances comply with the limit. This is showcased in Figure 2.4, where the medial axis of a shape with sharp features can be seen.

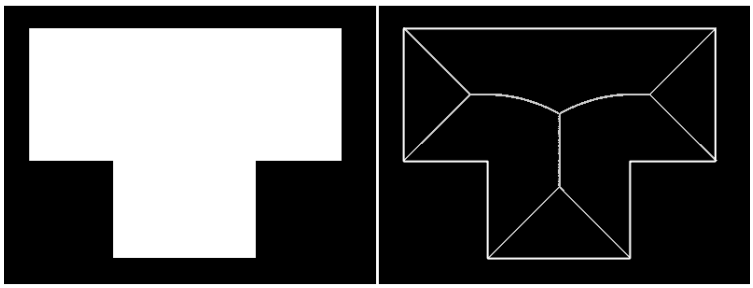


Figure 2.4: The medial axis representation of a shape with sharp corners. The medial axis is depicted as the skeleton like white lines within the outlined shape.

This approach is especially ill suited for measuring internal distance constraints for soft robots made using a single-piece mould, as the need for rounded corners would make the minimisation problem from Equation (2.2) unsolvable, since the mould would need overhang to model the smoothness of the corners.

Another approach for verifying that the internal structures are not too fine, is to use erosion and dilation. Whether a robot is thick enough can be verified by eroding a polyhedron using a spherical structural element - with radius δ corresponding to the minimum required thickness - and then dilated using the same structural element. If the polyhedron contained any features finer than the sphere with radius δ, they will not be present after the erosion and dilation. More formally, if erosion is defined as $\ominus$ and dilation is defined as $\oplus$, then a robot polyhedron $\mathcal{P}$ can be verified to have a minimum thickness of δ as,

$$(\mathcal{P} - \oplus_\delta(\ominus_\delta(\mathcal{P}))) \cup (\oplus_\delta(\ominus_\delta(\mathcal{P})) - \mathcal{P}) = \emptyset \tag{2.4}$$

However, again sharp features are lost by opening - i.e. eroding and then dilating - the polyhedron due to the usage of a sphere as structural element, as can be seen in 2D in Figure 2.5 below. In this example the structures violating the thickness constraints are indeed removed, but so are any sharp features that we wanted to preserve.

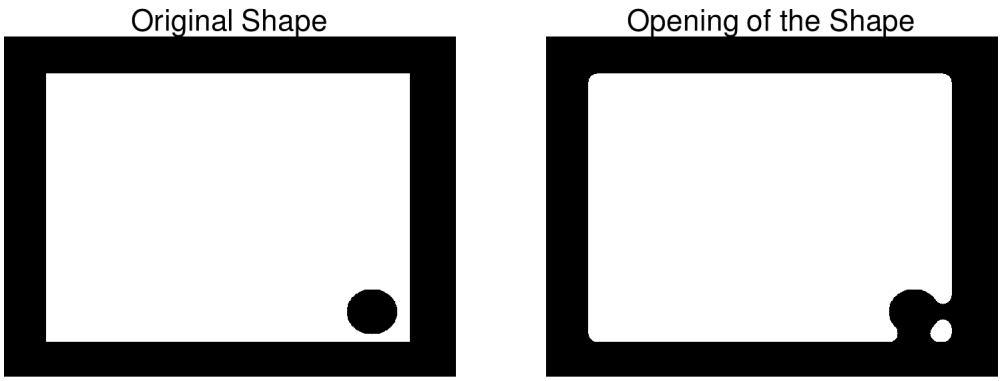


Figure 2.5: An example of how the opening of a shape - using a disk as structural element - is not equal to the original shape.

To efficiently test for sufficient thickness of a robot $\mathcal{R}$, we constructed a novel approach dubbed the *thickness cone* method. The method is related to the concept of a tangent cone known from optimisation [Tre83], and at a point, $\mathbf{p}$, represents all finite tangent directions with norm less than or equal to some *thickness* constant δ. The thickness cone can therefore be interpreted as the solid angle of the δ-ball with centre located at $\mathbf{p}$, intersected by the local surface $\mathcal{S}$ of the robot at position $\mathbf{p}$. See Figure 2.6 for examples of how the thickness cone will be placed on a surface $\mathcal{S}$.

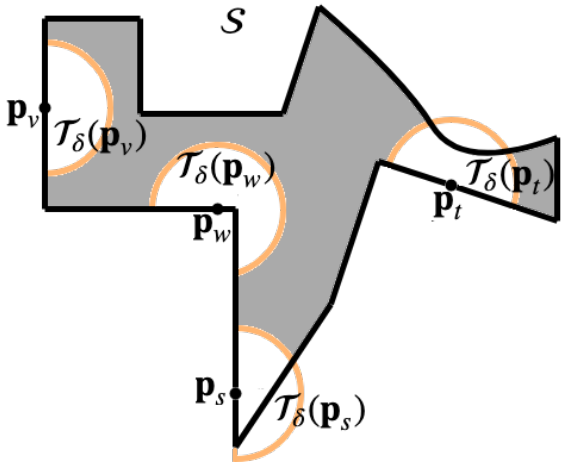


Figure 2.6: A 2D representation of a robot surface $\mathcal{S}$ with thickness cones at the points $\mathbf{p}_v$, $\mathbf{p}_w$, $\mathbf{p}_s$, and $\mathbf{p}_t$. Here the white areas represent the thickness cones.

Therefore, when measuring robot thickness, we actually want to verify that there doesn't exist any *backside* intersections between the thickness cone and the surface of the robot. As can be seen in Figure 2.6 only part of the thickness ball is defined inside the robot, and it is intersections between this 'inside part' and the robot surface that we define as backside or *true* intersections of the thickness cone. All other intersections happen from the outside of the robot, which means that they relate no information about the internal geometry, and we refer to these intersections as *false* intersections.

Now, let all points $\mathbf{p}$ of the surface of the robot be given by $\mathcal{S}$, and let the thickness cone with thickness δ at point $\mathbf{p}$ be given by $\mathcal{T}_\delta(\mathbf{p})$. Then, more formally, for a point $\mathbf{p}$, no points on the immediate adjacent surfaces will be part of the thickness cone $\mathcal{T}_\delta(\mathbf{p})$, since these can never create a 'backside' intersection. Observe in Figure 2.7 to the left a thickness cone which only has false intersections, and to the right a thickness cone with a true intersection.

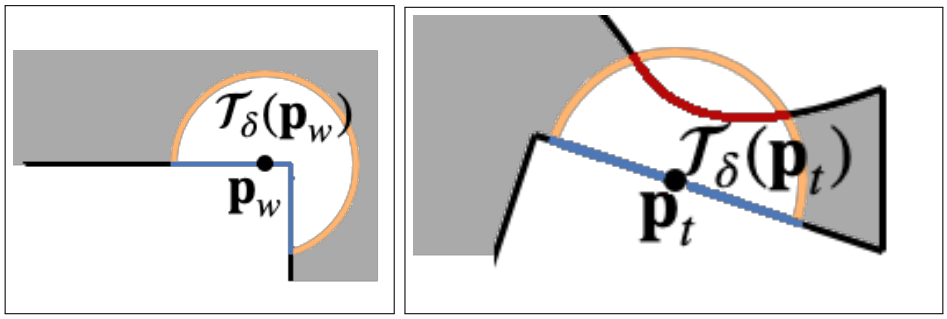


Figure 2.7: Left shows a thickness cone with only false intersections and right shows a thickness cone with a true intersection. The true intersection is marked in red and the false intersections are marked in blue.

We define the restricted surface neighbourhood $\mathcal{N}_\delta$ at $\mathbf{p}$ to be any point $\mathbf{q} \in \mathcal{S}$ that lies inside $\mathcal{T}_\delta(\mathbf{p})$, where a continuous path to $\mathbf{p}$ exists strictly inside $\mathcal{T}_\delta(\mathbf{p})$. The restricted robot surface at $\mathbf{p}$ is then simply $\mathcal{S}_\delta(\mathbf{p}) \equiv \mathcal{S}/\mathcal{N}_\delta(\mathbf{p})$. Thus, the thickness constraints can be stated formally as

$$\forall \mathbf{p} \in \mathcal{S}, \quad \mathcal{T}_\delta(\mathbf{p}) \cap \mathcal{S}_\delta(\mathbf{p}) = \emptyset \tag{2.5}$$

as any surface connected directly to $\mathbf{p}$ will always intersect with the thickness cone, this definition states the robot is only too thin if there exists any true intersections.

2.1.3 Surface Bending Constraints

From definition a polyhedron $\mathcal{P}$ is a piece-wise linear representation of a smooth surface. Returning to the original smooth surface, then from differential geometry we know that the curvature of a surface $\mathcal{S}$ is the product of its principal curvatures, or in other words, how much the surface bends in different directions. In the discrete setting of the polyhedron, we cannot measure the curvature directly, but we can easily compute the angles between its facets. As the angle between two facets increases in sharpness the robot can develop *fracture points*, which are points at which the robot is prone to rupturing. To satisfy the third robustness criteria we must make sure that no fracture points are created, and to verify that no fracture points exists, we typically examine the robot from a 2D viewpoint. Observe in Figure 2.8 an example of this view.

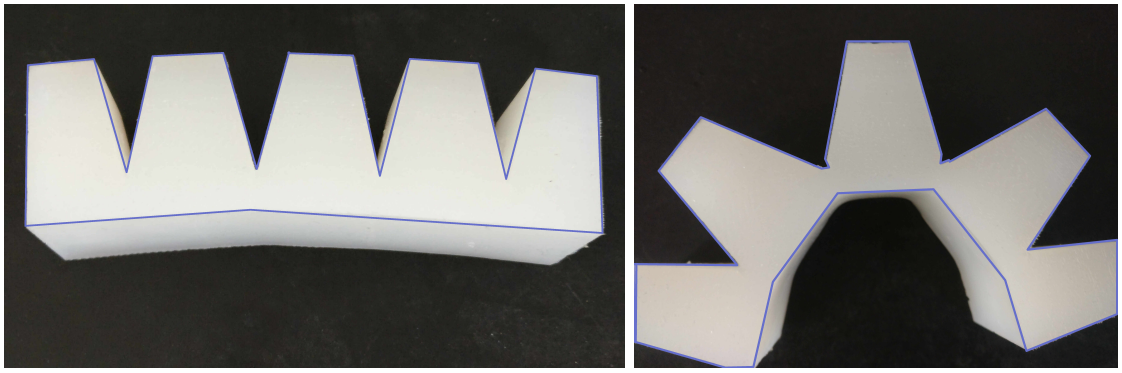


Figure 2.8: A soft robot with the edges of a facet highlighted before and during actuation.

Now, in a 2D setting the discrete curvature between two adjoining edges is given by $K = 2\sin\left(\frac{\theta}{2}\right)$ where θ is the dihedral angle between the normals of edges [Cra18]. To satisfy this robustness condition, we therefore need to make sure that the angles between facets do not become too steep. We can then state the curvature constraint of a robot for any adjacent normals $\mathbf{n}_i$ and $\mathbf{n}_j$ along a shared concave edge as,

$$\cos^{(-1)}\left(\mathbf{n}_i \cdot \mathbf{n}_j\right) \geq \theta \tag{2.6}$$

The curvature constraint is important to make sure that the robot doesn't become vulnerable to fracturing in the concave parts of the robot. However, for the convex angles of the robot, vulnerable low mass areas should not be allowed to exist as the robot design would otherwise be invalidated by the thickness test.

2.1.4 Cable Embedding Constraints

In cable-driven soft robots deformations of the soft body are achieved by pulling or relaxing a cable. As a cable interacting directly with the silicone can damage the soft body over time, some sort of cable shielding is usually added to the robot. In practice, the tubes used are often very light and consist of another material than silicone. This means that they can buckle if bend too far, which can make cabling impossible. Hence, the curvature of the cable shielding needs to be constrained in order to ensure a safe manufacturing process. To provide smoothness and local support we choose to model cable shielding using non-periodic non-uniform k-order B-splines [Erl+05], which means that we describe each cable by its arc-length parametrisation. Given the B-spline representation, $C(u)$, we can then easily compute the curvature of the spline, as it is given by its second-order derivative. Thus, to give a strong guarantee on the feasibility of a design, each cable $C(u)$ must satisfy,

$$C''(u) \leq \kappa \tag{2.7}$$

where κ is the maximum amount the cable can bend before buckling. This condition only ensures that the sheathing does not bend during fabrication, and a more thorough approach need to be taken into account to also apply the buckling constraint during actuation of the robot.

Besides from the buckling constraint, one should verify that all of the cable shielding will be placed *inside* of the robot to ensure tunnelling is in a proper configuration - this is done by visual inspection for now.

As the cable tunnels of the robot will often create very low density areas of silicone - see Figure 2.2 - the thickness constraints from Equation (2.5) need to be validated on the robot after tunnelling has been added. However, we cannot use the same model - i.e. the model with tunnels - for validating the mould extraction feasibility constraints from Equation (2.2), as the opposing normals within the cable tunnels would make a mould extraction direction unable to exist. Thus, we need two representative models of the robot; one with cable tunnelling and one without.

In order to make a strong feasibility guarantee for a robot $\mathcal{R}$ with facets $\mathcal{F}$, surface $\mathcal{S}$, cabling $\mathcal{C}$ and extraction direction $\mathbf{d}$, it must be satisfied that,

$$\begin{aligned} \mathbf{d}_x\mathbf{n}_x + \mathbf{d}_y\mathbf{n}_y + \mathbf{n}_z \leq 0, &\quad \forall \mathbf{n} \in \mathcal{P} \\ \mathcal{T}_\delta(\mathbf{p}) \cap \mathcal{S}_\delta(\mathbf{p}) = \emptyset, &\quad \forall \mathbf{p} \in \mathcal{S} \\ \cos^{(-1)}(\mathbf{n}_i \cdot \mathbf{n}_j) \geq \theta, &\quad \forall \text{ adjacent } (\mathbf{n}_i, \mathbf{n}_j) \in \mathcal{F} \\ C''(u) \leq \kappa, &\quad \forall C(u) \in \mathcal{C} \end{aligned} \tag{2.8}$$

which is the geometric constraint model resulting from the combination of all the robustness conditions. In the pre-project it was shown that it is possible to experimentally find values for δ and θ that ensured safe manufacturing. The specific values found can be seen in Table 2.1.

Parameter	Value
Minimum thickness at any point	0.5 cm
Minimum angle allowed between facets	20°

Table 2.1: Recommended settings from the pre-project for manufacturing soft robots using EcoFlex 00-50 [Smo], which strongly guarantee that the robot is manufacturable.

However, no buckling constraint value, κ, was found during the pre-project as all cables were

assumed to be unable to bend. We will also make this assumption in this thesis, but in future work it would be beneficial to implement a numerical buckling verification.

Cable Shelves

Finally, a practical problem we encountered during experimentation with different robot designs is that the addition of cable shielding creates overhang that makes the mold extraction problem unsolvable, and causes the robot to be difficult - if not impossible - to remove from its mold. To solve this problem we have chosen to add what we refer to as cable shield 'shelves', whenever a cable needs to be placed inside the mold. Observe in Figure 2.9 a cut-out of a robot mold with a cable shelf placed to make sure that there will be no conflicting normal directions.

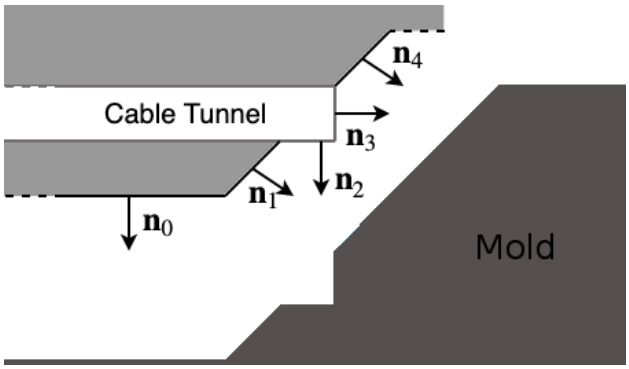


Figure 2.9: Cable shielding is embedded into the robotic design (grey) to create shelves in the mold (dark). The shelving strategy ensures that the cured robot with embedded cable shielding can be extracted from the mold.

2.2 The Manufacturing Process

As mentioned previously, the most common way to create a soft robot is to begin by designing it using a CAD tool. We will be using the design tool created for the pre-project, which means that we begin by creating a digital 'positive' of the soft robot using CSG. In other words, we create the ground truth that we aim to manufacture. Once this model has been created, we then create a mold - in other words, the 'negative' - with a single cavity corresponding to the soft body we wish to produce. Observe in Figure 2.10 an example of a soft robot positive and negative.

Figure 2.10: The positive and negative of a soft robot.

To verify that the robot can be manufactured, we must ensure that it satisfies the robustness criteria from Equation (2.8). If we can strongly guarantee that the robot is manufacturable, we

then begin preparing it for 3D printing. Using the Cura [BVa] software we create a `gcode` file containing the 3D printing parameters, which determines how the model will be printed in terms of layer height, infill density, etc. For this thesis, all molds will be created using an Ultimaker 2+ 3D-printer, which is also the printer for which Cura was developed.

Once the mold has been created, it is prepared for casting by removing small imperfections in the plastic, adding a layer of Mold-Release [Inc18] to ease the extraction process, and finally, adding actuation devices such as cable sheathing. Afterwards, the material which will make up the soft body is mixed and treated to remove air-bubbles from the mixture. For this thesis, we only use silicone elastomer for the outer frame of the soft robots, and for each material the process of readying the silicone differs a little. To remove air-bubbles we use a vacuum chamber and once all air has been removed from the mixture it is poured slowly into the mold. Observe in Figure 2.11 a 3D printed mould in the process of having silicone poured into it.

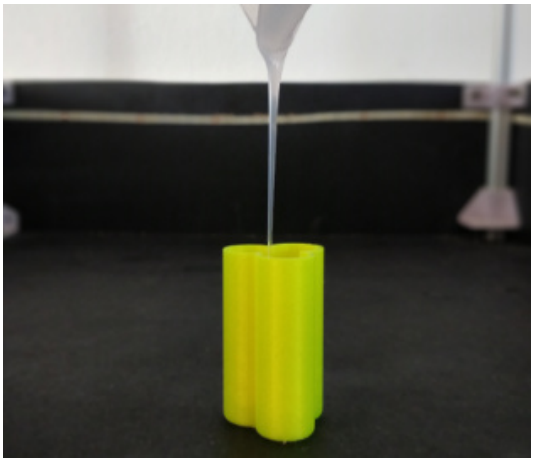


Figure 2.11: The 3D printed mould having silicone poured into it.

When the silicone has cured, the soft robot can be removed from its mold. If the model passed the robustness criteria, there is a strong guarantee that the robot can be removed from the mold without rupturing. However, due to the nature of silicone pulling near thin areas of the soft body can still cause rupturing if done excessively. Observe in Figure 2.12 from left to right the different stages of the manufacturing pipeline described above.

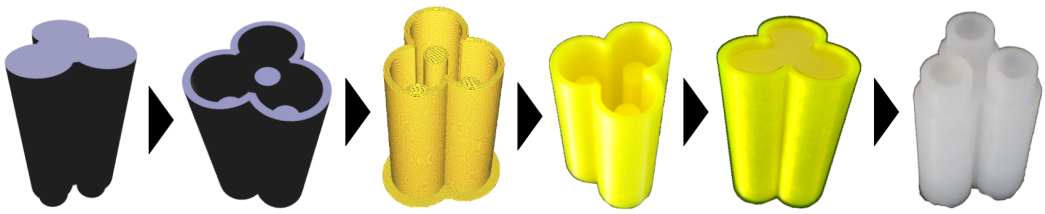


Figure 2.12: The different steps of the manufacturing pipeline.

Once a soft robot has been fabricated, we can begin verifying that it exhibits the expected behaviour. However, currently, if we need to re-iterate on the initial design, we have to repeat all steps of the manufacturing process.

Chapter 3

The Learning Platform

As soft robots interact with their environment by deforming under actuation - as opposed to robots consisting of rigid structures - their kinematics do not only depend on geometry, but also on the material properties of the soft bodies. This makes it difficult to construct an analytical model that can describe their mechanical behaviour accurately [CED17]. In this thesis we will be controlling the soft robots using a predefined control policy learnt through simulated data. To verify that the robots behave as expected, we will be using an existing platform for collecting data of the motion of the robots. The platform - named the *Learning Cube* - was built as part of the research performed in [Hol18], which aimed to create a versatile and convenient way of learning the inverse kinematics of an arbitrary soft robot. We will, however, only use the platform to examine the gap between our simulations and reality.

3.1 Components of the Platform

The Learning Cube platform consists of five main components:

1. The soft robot we will be actuating.
2. The actuation system utilised by the robot.
3. The rig which the soft robot is mounted on.
4. RGB-D sensors that measure the deformations of the soft body.
5. The `pySoRo`[HE18] software, which enables the collection and processing of data, as well as a way to control the actuators.

In this chapter each component will be described in detail, and in Figure 3.1 an overview of the physical components of the platform can be found.

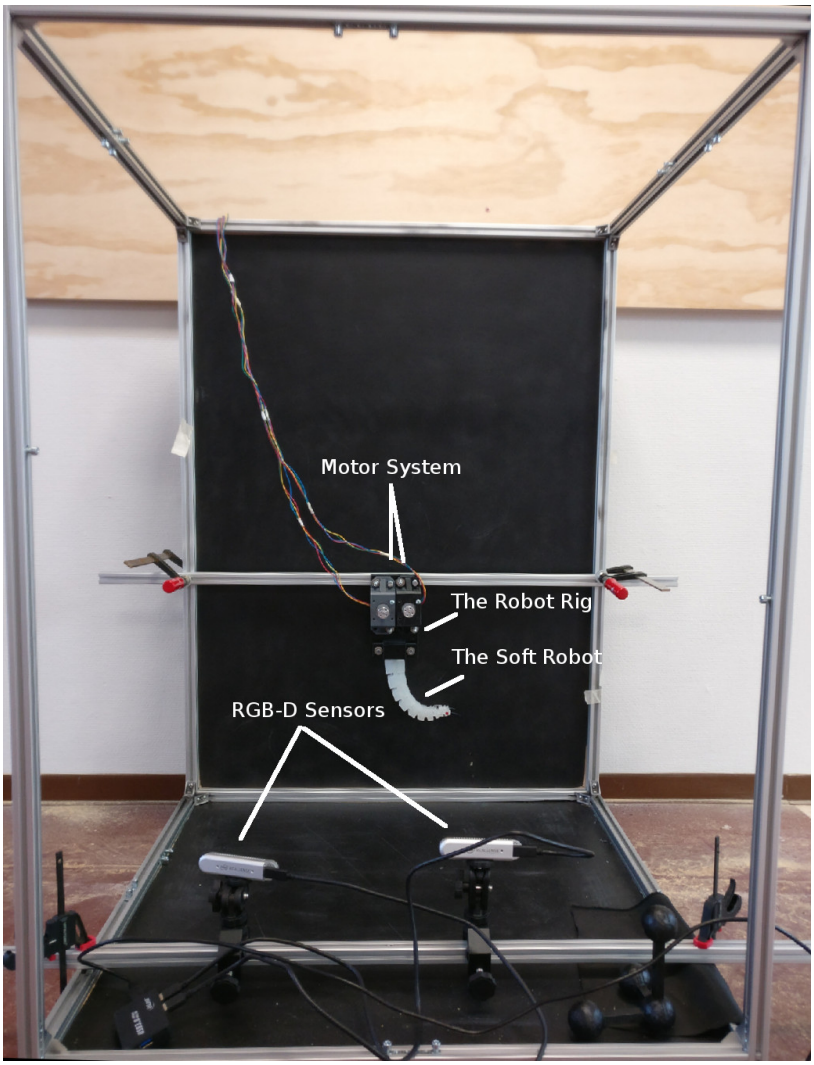
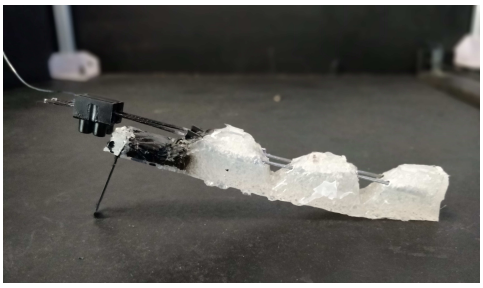


Figure 3.1: The physical components of the Learning Cube.

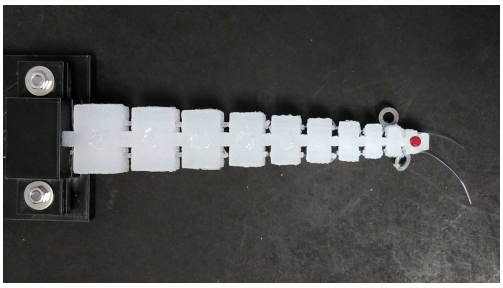
The Soft Robots

As mentioned previously, soft robots consist of a soft frame or shell and are actuated by some kind of motor system connected to the soft body. Popular actuation systems for soft robots include pneumatic networks, cables connected to small motors and Shape Memory Alloy (SMA) coils shaped like springs, that can extend and contract when heated. We note that impressive robot designs have been proposed utilising SMA coils [LLT11], but due to availability and difficulty of use, we will not be working with SMA coils in this thesis. Instead we focus on cable-driven and pneumatically actuated soft robots, as these suit our manufacturing capabilities better.

The Learning Cube uses a *marker based* approach to figure out the configuration of the robot, which means that the robot needs to be equipped by a number of markers that can be segmented from the rest of the soft body. This approach accommodates the variance in the design of soft robots well, as the only requirement placed on the design is that it must be possible to place markers on the soft body. An example of two robots utilising the same actuation system, but with vastly different designs can be found in Figure 3.2.



(a) The soft robot originally built for usage in [Hol18].



(b) One of the robots which will be used in this thesis (Robot A.2).

Figure 3.2: Examples of soft robots that can be used with the Learning Cube.

The Learning Cube is not limited by the actuation system of the soft robot placed inside of it either, which means that the platform is not limited to specific robot designs. Instead, all that is required by the Learning Cube is that the actuators are controllable through a digital signal - in other words, the control parameters of the robots must be controllable from the system running the `pySoRO` software. The control parameters that map to robot configurations are the measurable forces applied to the actuators of the robot. For cable-driven robots this could be the amount of degrees a stepper motor has turned, and for a pneumatically actuated robot the control parameters could be the air pressure in the air chambers of the robot.

Cable Driven Robots

The majority of the robots used in this thesis are cable-driven, simply because we have more experience constructing those types of robots. To control the length of the cables actuating the robot, we employ 2-phase motors from `SparkFun Electronics` [Ele19c] . The motor is a stepper motor which takes 1.8 degree steps, and therefore, the motor can do 200 steps in one complete revolution. The motor shaft - which is where the cable is fitted - has a radius of 2 mm, which means that one step can tighten or loosen the cable by 0.0625 mm, and one revolution corresponds to changing the length of the cable by 12.5 mm. The motors used for controlling the soft robots are connected to `stepper motor drivers` [Ele19a], that are used to control the speed, acceleration and absolute position of the shaft. The motor set-up can be found in Figure 3.3.

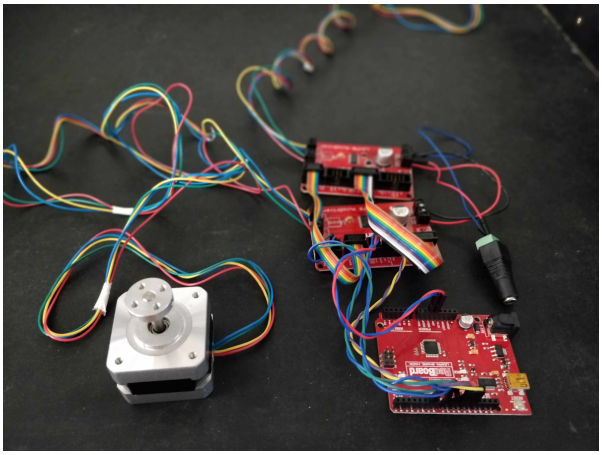


Figure 3.3: The stepper motor set-up.

The motors employed are all daisy chained to a `SparkFun RedBoard` [Ele19b] programmed with `Arduino` [Ard], which can be controlled by sending and receiving messages over USB, between a computer and the board.

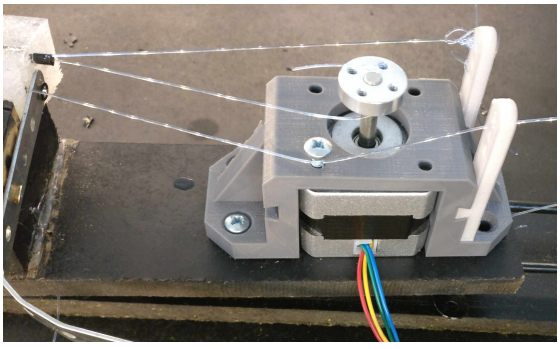
Pneumatically Actuated Robots

In terms of pneumatic actuators, we were unfortunately unable to acquire a system which we could control digitally. Therefore, we will be using a human operator to power a series of syringes, that will function as pistons. An example of such a syringe can be found in Figure 3.4.

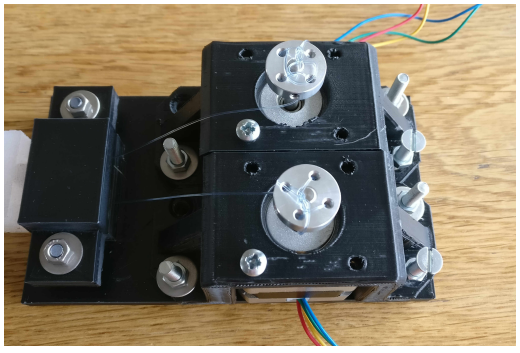
Figure 3.4: The 'pneumatic actuator' we will use to power our pressure-driven robots. One syringe will be used for each air-chamber, and can increase the volume inside an air-chamber by 120 ml.

Robot Rig

To keep our robots in place and allow for the placement of actuators close to the robots, we will be using 3D-printed custom rigs. An example of some of these 3D-printed rigs - with two motors attached - can be seen in Figure 3.5.



(a) Robot Rig B.1, which was originally built for usage in [Hol18].



(b) Rig B.2, created for use with Robot A.2.

Figure 3.5: An example of different rigs on which robots can be mounted.

RGB-D Sensors

For this thesis, up to two `Intel RealSense Depth D415` cameras [Cor19a] - that can capture RGB images as well as depth - will be used for collecting data. However, for some of our experiments, a single camera will suffice. To work with these cameras, the `Intel` provided `librealsense` software [Cor18] have been used to achieve both depth and colour streaming, as well as a synthetic stream for acquiring point clouds of the scene.

Figure 3.6: The `Intel RealSense Depth D415` camera.

The Cube

The cube itself consists of an aluminium frame and a wooden box, which is painted black to make segmentation of the RGB-D data easier. The cube measures 74 cm $\times$ 74 cm $\times$ 100 cm, and the `Intel RealSense Depth D415` cameras can be mounted anywhere on the aluminium frame. This makes it easy to adjust the cameras to suit a specific robots shape and size. The rig containing the soft robot and its motors can be placed anywhere inside the wooden box, but to make sure that the cameras do not capture too much of the background outside of the box, it should be placed near the centre.

`pySoRo`

To use the data captured by the RGB-D sensors, we will be using the `pySoRo` software package. This software package contains methods to calibrate the sensors, interfaces to communicate with the `Arduino` boards, tools needed to acquire the RGB-D data in a systematic manner and the code needed to find and segment the makers placed on the soft robot. We will not go into details as to how this software works, but instead refer to the thesis [Hol18], for which the software was developed.

3.2 Collecting Data Using the Learning Cube

Sensor Calibration

Before we can begin collecting data observations of the motion of a soft robot, we need to calibrate the RGB-D sensors. We want to acquire a set of 3D coordinates corresponding to the positions of the markers placed on the soft robot. To find these positions, everything but the markers are segmented away, which means the cameras need to be adjusted such that the segmentation will be as noiseless as possible.

The cameras we used were calibrated manually using the `Intel RealSense Viewer` [Cor18] going by the rule of thumb that when a white ball was placed in the empty box, it should be difficult to discern any details of the wooden box, while the white ball should remain sharp and

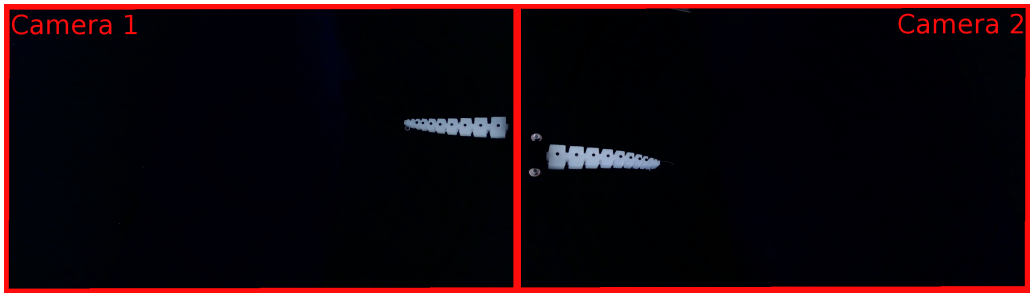
not blurred. In Figure 3.7, an image of the white balls can be seen before and after calibration for one of the cameras.

(a) Before calibration.

(b) After calibration.

Figure 3.7: Images of the white balls placed inside the cube, before and after calibration.

We also need the markers to be partially visible at all times, but this might not be possible for all robot configurations using a single camera. If the image is noisy or some of the markers are hidden, we need to estimate their positions. Observe in Figure 3.8a an example where all markers of a robot are visible to both cameras, and in Figure 3.8b an example of all markers only being visible from one of the vantage points.



(a) A configuration where all markers are visible to both cameras.

(b) A configuration where all markers are not visible to Camera 1.

Figure 3.8: Two different configurations showcasing the necessity of two cameras.

Post Processing of Camera Calibration Data

As can be seen in Figure 3.8, we might need to interpolate marker positions. Therefore, once the sensors have been calibrated, we need to find a mapping between the coordinate systems of the

two cameras. In other words, we need to find a rotational matrix $\mathbf{R}$ and a translational vector $\mathbf{T}$, that satisfies for all $\mathbf{P}$ that,

$$p^A = \mathbf{R}p^B + \mathbf{T}, \qquad \forall(p^A, p^B) \in \mathbf{P}$$

where p^A is the position of a marker in the coordinate system of camera A, while p^B is the position of a marker in the coordinate system of camera B.

To construct the mapping between cameras, we will be using the `pyCALIBRATE` part of the `pySoRo` software package. This software computes the rotational and translational components by comparing the positions of known points in each coordinate system. We acquire these points by placing five or more white balls into the Learning Cube, capturing colour, depth and pointcloud data for each camera, and then repeating this with the balls removed. It is not necessary to remove the rig and robot from the cube during calibration - as long as their positions do not change - because they will then be part of the background which can easily be removed from the image. An example of how these balls can be placed can be seen in Figure 3.9.

(a) The Learning Cube before the balls are placed.

(b) The Learning Cube with white balls inserted.

Figure 3.9: The Learning Cube with and without white balls placed into it.

This approach makes it easy to segment out the white balls, and therefore, the centroids of the balls serve the purpose of known points in both coordinate systems well. It is important that the balls are placed such that the configuration of the balls looks different from the vantage points of the different cameras, as there might otherwise exist multiple mappings between the two coordinate systems.

Robot Motion Data Acquisition

When the mapping between the coordinate systems of the cameras have been found, we can begin sampling the motion of the soft robot. Depending on the purpose of the motion data, we might choose to capture images of the robot in a set of predefined positions or instead capture data after each action of a control policy is performed. The cameras are not particularly well suited for continuously capturing images, due to a bug in the `librealsense` software, making 2-dimensional data protocols unavailable. Therefore, the actions performed might need to be broken into smaller sub-procedures between which images can be captured.

For a robot, such as Robot A.2, the range of motion is relatively large, due to the robot being actuated by two cables, as can be seen in Figure 3.10.

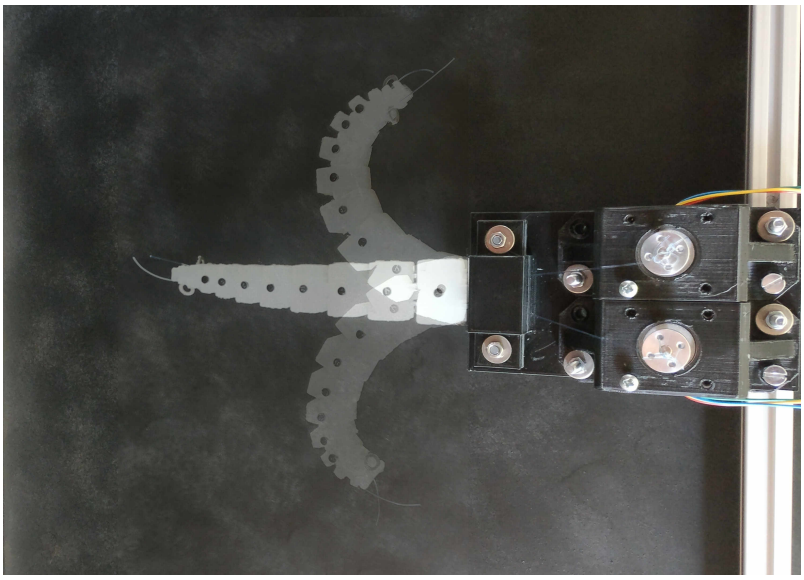


Figure 3.10: Range of motion for Robot A.2.

The Learning Cube is well-suited for validating simple learning tasks, but might be a poor fit for validating more complex policies due to the limitations of the cameras. Especially continuous tasks are difficult to perform and validate, as it might not be possible to acquire information about the state of the robot and its surroundings quickly enough to react to environmental changes. However, as long as the task can be broken into smaller subtasks - and time is not critical - we should be able to capture the data needed to measure the real world performance of the policy.

Post Processing of Motion Data

The configuration of a robot is represented by the positions of the markers placed on it. The markers need to be segmented from the RGB-images of the soft robot and mapped to 3D-coordinates using the pointcloud data. This is the final step of the data collection pipeline, and the resulting marker positions can then be used for validating experimental results.

In the `pySoRo` software package, the post processing pipeline begins by converting an RGB image as the one in Figure 3.11a into its HSV representation.

(a) An RGB image of a soft robot within the cube taken by one of the cameras.

(b) The HSV representation of the image from Figure 3.11a.

Figure 3.11: An RGB image of a robot next to its HSV representation.

The image is converted to HSV, because the saturation (S) and hue (H) channels of the image can then be discarded, making the segmentation process easier. The robot can then be segmented from the background by performing a threshold of the value channel, and due to the contrast between the robot body and the markers, this method provides a good segmentation in most cases. Observe in Figure 3.12 the value channel of the HSV image in Figure 3.11b, and the segmentation resulting from performing a threshold of the value channel.

(a) The value channel of 3.11b.

(b) The segmentation resulting from thresholding the value channel from 3.12a.

Figure 3.12: The value channel of the image from Figure 3.11b, and the segmentation resulting from thresholding the value channel.

Once the body of the soft robot has been found, the markers are then discovered by finding 'holes' in the robot body. We find the holes by filling the holes and then substracting the 'filled' robot body from the original segmentation. Performing these two steps for the segmented body from Figure 3.12b can be seen in Figure 3.13.

(a) The segmented body from Figure 3.12b with holes filled.

(b) The result of subtracting Figure 3.13a from Figure 3.12b.

Figure 3.13: The marker candidates for the robot from Figure 3.11a.

Now, all that remains are the 'holes' of the original image. We always know the exact number of markers on the soft body, and we can therefore filter out some of the wrong marker candidates by removing the smallest candidates until only the correct amount of markers remain. In Figure 3.14 the original image of the robot can be seen with the marker positions found by the post processing pipeline marked in blue.

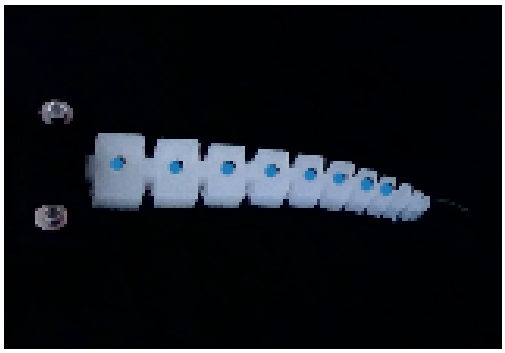
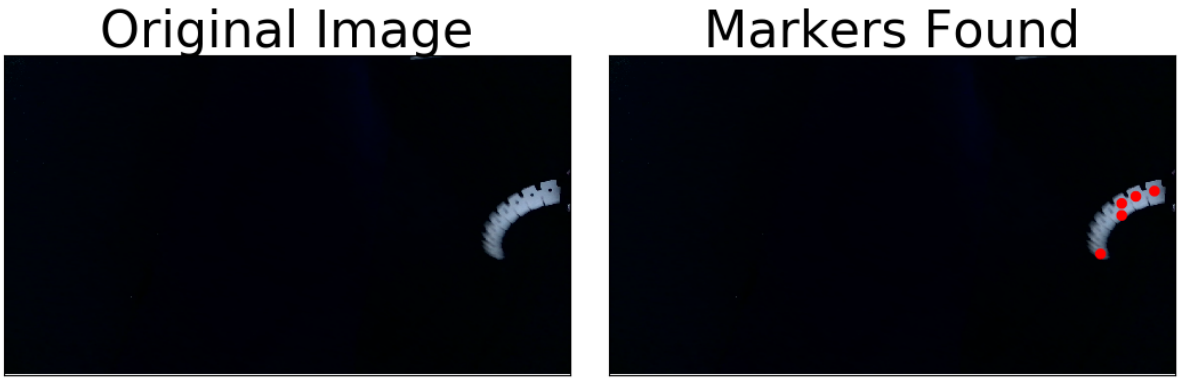


Figure 3.14: The final marker positions found for the robot from Figure 3.11a. The markers are highlighted in blue for improved visibility.

The segmentation is not guaranteed to succeed, and in some instances the markers cannot be found due to either being placed too close to the boundary of the soft body or the image being blurry. Observe in Figure 3.15 an example of a configuration where all the markers cannot be found.



(a) Blurry image captured by a camera.

(b) Markers detected highlighted in red.

Figure 3.15: An example where the segmentation scheme cannot find all markers. Notice that the blurriness of the image causes the markers to blend into the body of the robot, making it difficult to create a good segmentation.

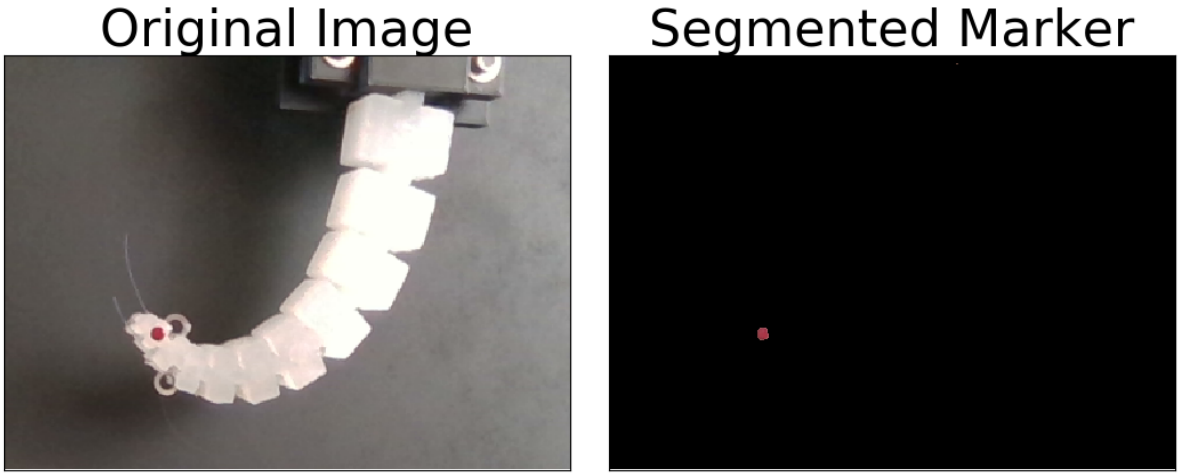
As mentioned previously, we will overcome the challenge of markers not being visible at every step of the data acquisition, by using the `pySoRo` software package to interpolate missing marker positions between frames. The interpolated markers are of course not entirely accurate, and we therefore aim to capture images of the soft robot from which we can extract all marker positions, as this will produce the best data for validating a control policy.

3.2.1 Increasing the Accuracy of the Segmentation

The segmentation scheme described above rely on contrast between markers and the soft body of the robot. As can be observed in Figure 3.15, it becomes difficult to place markers that are visible as 'holes' in certain configurations, when the geometry of the robot is very fine. Instead of using contrast to segment markers, we will be applying a colour threshold approach.

We proceed as for the original segmentation scheme by transforming the RGB images into their HSV representations. Now, for each of the marker colours used we will create an upper and lower range corresponding to HSV representation of that colour. The markers can then be found by using the colour ranges as masks, which will leave only pixels of the desired colour in the images.

In HSV, colours can typically be defined to lie within two or more ranges of saturation, hue, and value, combinations. For the colour red, for example, these ranges are given by $(0 \rightarrow 10, 100 \rightarrow 255, 100 \rightarrow 255)$ and $(160 \rightarrow 180, 100 \rightarrow 255, 100 \rightarrow 255)$, and using the combination of these masks would result in only red pixels remaining in the image. Observe in Figure 3.16 an example of the colour segmentation applied to a robot with a single red marker placed.



(a) Original RGB image captured by one of the cameras.

(b) Everything but the marker removed from the image.

Figure 3.16: An example of how markers are found using the colour segmentation scheme. The images have been zoomed in to increase the visibility of the marker. Observe the large variance in the lighting in the RGB image.

The colour segmentation scheme is weak when the background of the image can contain colours in similar ranges as the markers of the robot. However, as we are in total control of the background of the cube, the colour of the soft body, and, of course, the colours of the markers, this vulnerability will not be relavant. On the other hand, the contrast based segmentation scheme does not work when the background of the box reflects any light, which means that it needs to be placed in a very controlled environment to function properly.

3.3 Experiments

3.3.1 Capturing Robot Motion using the `pySoRo` Segmentation Scheme

Introduction

We want to investigate if we are able to capture the motion of a robot inside the learning cube using the pre-existing segmentation scheme described in Section 3.2. In other words, we want to know if we can reliably acquire the positions of the markers in images taken of the robot during actuation.

Procedure

For this experiment two `Intel RealSense Depth D415` cameras were used to capture images of the robot in different configurations. In total thirty images was taken by each camera, resulting in sixty images in total, and for this experiment the robot was equipped with five markers. To measure the accuracy of the marker segmentation, all of the images were also marked manually to create a 'ground truth' segmentation. Observe in Figure 3.17 how such a marking was performed.

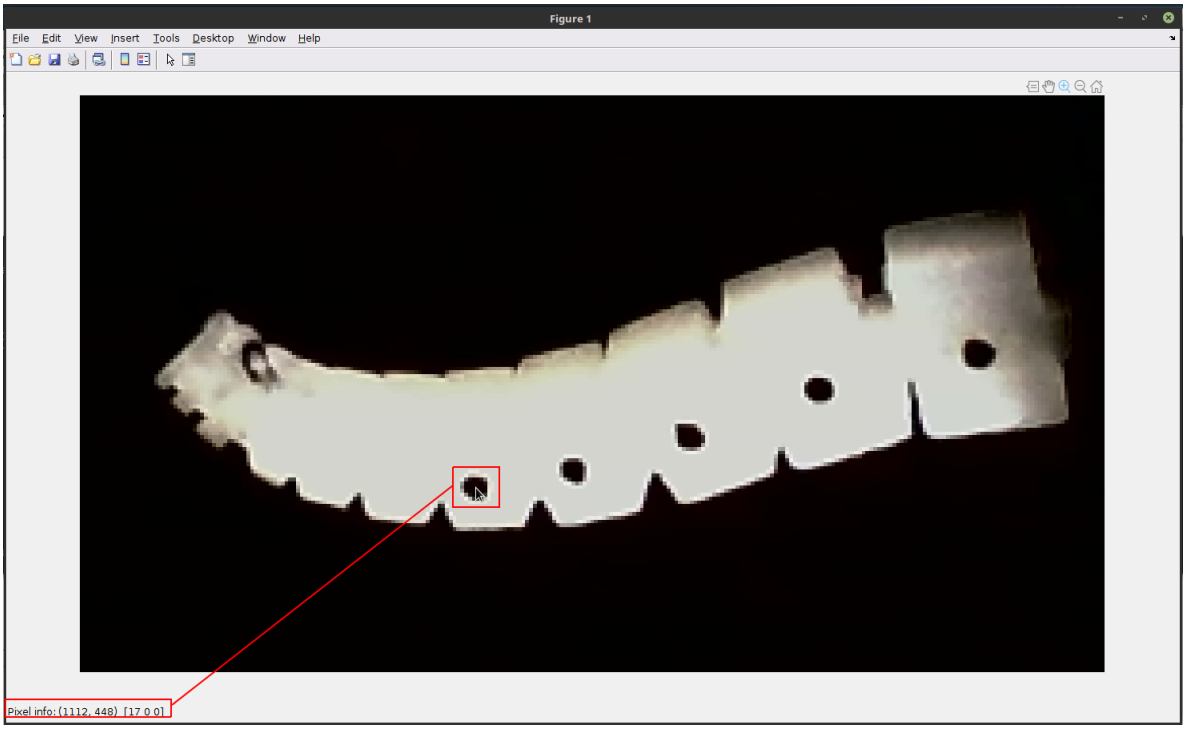


Figure 3.17: The centroid of each marker was manually found using the `MATLAB impixelinfo` functionality.

The accuracy of the segmentation will therefore be judged based on the mean distance (pixel-wise) between the centroids of the `pySoRo` segmentation and the manually segmented markers. We will also measure the accuracy of the segmentation in terms of missing markers, since we ideally want all markers to be found in each image.

Results

Observe in Figure 3.18 a scatter plot with manually found markers in orange and the segmented markers in blue.

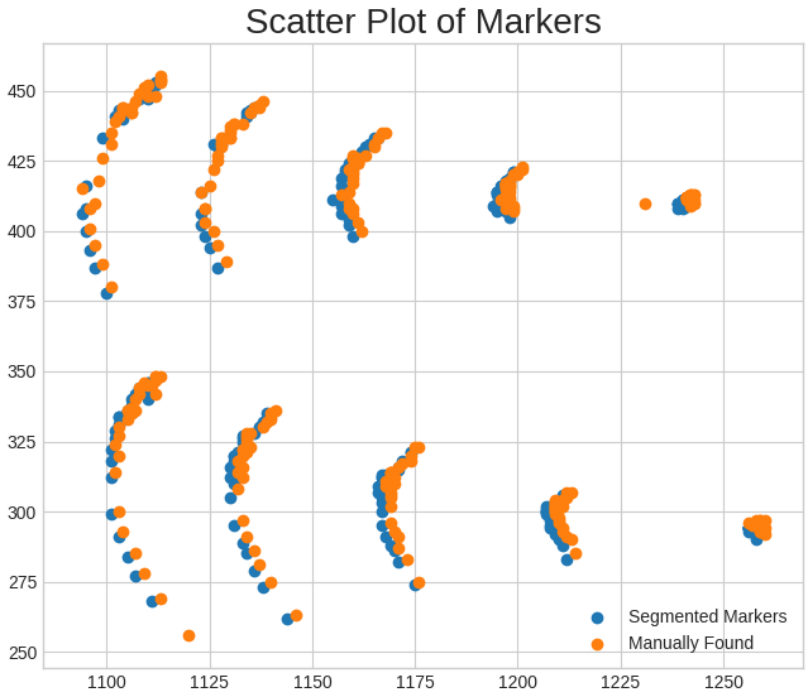


Figure 3.18: The segmented and manually found markers plotted in the same coordinate system.

We observe that there are significantly less blue than orange markers, but we also notice that those found are very close to the ground truth. In Figure 3.19 the ratio of markers found vs. markers missing can be found.

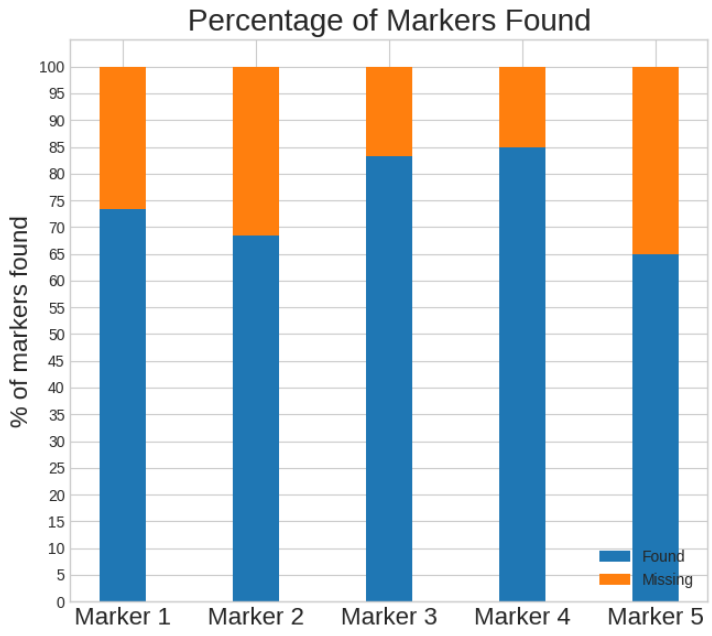


Figure 3.19: Ratio of markers seen (blue) against markers missing (orange) for each marker.

The markers are numbered such that the marker closest to the tip is 'marker 1', and 'marker 5' is the marker closest to the base of the robot. Notice that the number of missing markers are the worst closest to the base and the tip, and observe in Figure 3.20 an example of an image where the last marker could not be identified due to poor light conditions.

Figure 3.20: Poor light conditions making the extraction of every marker centroid difficult.

Observe in Figure 3.21 the mean distance between each marker and its ground truth plotted along with the standard deviation.

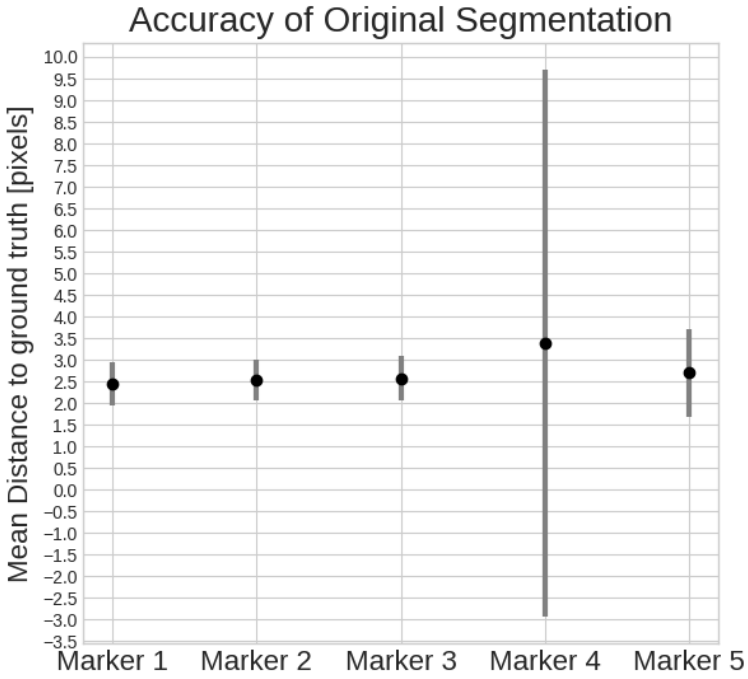


Figure 3.21: Mean and standard deviation in terms of pixel-wise distance to ground truth markers.

If we examine the mean pixel-wise distance for each of the markers to its ground truth counterpart, we see that it is actually the fourth marker that displays the largest variation between segmented and manually found positions. These result indicate that a segmentation approach

based on contrast does not produce perfect results, as certain configurations mixed with variance in light conditions can lead to a poor percentage of ratio of markers being found. In the `pySoRo` software package, this is mitigated by an interpolation scheme which estimates the positions of missing markers based on their current trajectory. However, ideally we want our captured data to have all markers present as this will allow us the best starting point in terms of validating a control policy.

3.3.2 Capturing Robot Motion using the Colour Segmentation Scheme

Introduction

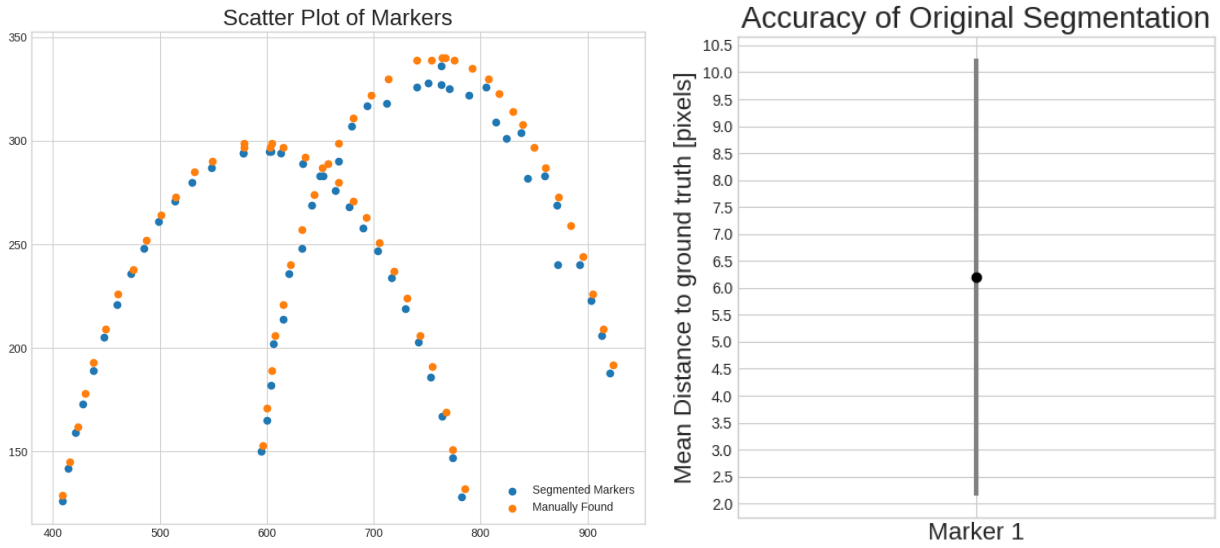
In this experiment we want to assess the effectiveness of the colour segmentation scheme described in Section 3.2.1 in terms of capturing robot motion.

Procedure

As in Experiment 3.3.1 we proceed by capturing images of the robot in different configurations, and manually find marker centroids. For this experiment we also captured sixty images of the robot, but this time the robot was only equipped by a single red marker at its tip - i.e. at one of the positions where we saw a large ratio of missing marker data in Experiment 3.3.1. We measure the accuracy in the same manner as in the previous experiment in terms of a visual inspection of a scatter plot, and the mean and standard deviation of pixel-wise distances between the manually found markers and the markers located by the segmentation scheme.

Results

Observe in Figure 3.22a a scatter plot of both markers found by the segmentation and the ones manually located, and observe in Figure 3.22b the mean and standard deviation in terms of pixel-wise distance to ground truth markers.



(a) Scatter plot of the markers.

(b) The mean and standard deviation in terms of pixel-wise distance to ground truth markers.

Figure 3.22: Result of the colour segmentation experiment.

For the colour segmentation scheme, we see a slightly larger variance in mean distance to the ground truth than for most of the markers in Experiment 3.3.1. In terms of the scatter plot, we overall see a good trend in terms of the markers found by segmentation lying in very close proximity with the ground truth markers, except at the top of the motion for one of the cameras. However, in all of the sixty images the marker was found. In future experiments, we therefore need to carefully consider which of the segmentation approaches we want to apply. The success of a segmentation scheme is greatly affected by variables such as the light condition and the settings of the cameras, and for our particular set-up, the colour-based segmentation approach seems to be the most reliable.

Chapter 4

Cable Actuation

Soft robots are often designed to mimic biology. Common sources of inspiration include octopi [Gal+16], geckos [Gli+18] or parts of the human body [Too18], due to their natural versatility to variable tasks. For the latter type, the robots are often designed to be cable-driven because the cables of the robots resemble tendons connecting muscles to bones. We want to be able to simulate the behaviour of cables actuating soft bodies, since this would allow us to evaluate and train policies for our cable-driven robot designs before performing the time-consuming manufacturing task. Simulating the behaviour of cables is a well researched topic, and there seems to be two notable ways in which the simulations are performed. Either the cables interacts with other objects behaving like a cable would in the real world - i.e. create frictional forces, contact forces, etc. - or else the cable is only an abstraction interacting solely with the object it is embedded within.

In [Zha+19] a motion control scheme for a cable-driven catheter robot was presented. Here, cables were discretised as points on a mesh - i.e. along mesh nodes where a real cable would make contact with the body of the robot - in which the forces created by the cable where modelled. To control the robot the authors relied on a set of contact points, and to find the necessary cable forces to achieve a certain configuration, an inverse kinematics problem was solved. The idea of controlling cables by an inverse kinematics approach was also present in [Meg+17], where cables were used to control animatronic puppets. In this work, the robot was controlled through torque created by cables pulling on the different joints of the robots. The cable forces required to achieve a certain robot configuration were similarly found by solving for the actuation forces - and joint angles - that resulted in an equilibrium state in terms of torque.

In terms of soft robots, [BKC17] have also shown some promising work in terms of controlling cable-driven robots. The authors presented 'plush' robots, where cables were modelled as *tendons*, which were considered as piecewise linear curves connecting nodes on a mesh. In [BCC], the cable model was further elaborated on, with the contraction of tendons being modelled as changing the rest length of its underlying *unilateral spring model*. This means that these types of cables are controlled by lowering the initial length of the cable, which corresponds to using a motor to reel it in, while a negative change to the initial length would correspond to making the cable 'slack'. These works also aimed to solve an inverse kinematics task, which means that the control scheme is a bad fit for our objective. The intention of our simulations is to interactively explore the workspace of our robots, whether the goal is to learn a control policy or simply testing the limitations of the robot design. Therefore, we want to create a model that given a control parameter produces a configuration - and not the other way around.

The idea of modelling cables as tendons seem to be a good fit for soft robots, as many designs aim to emulate biology. In [Eng12] a somewhat similar approach to modelling cables for simulating active contractions of deformable models was presented. Instead of using piecewise linear segments to model tendons, the concept of *activation splines* was introduced. An activation spline represents the activation forces in a deformable model using a spline structure, which has the advantage over the piecewise linear segment representation, that forces can be found for any point along the spline. However, in terms of discretisation, a discretised activation spline resembles the tendon representation from [BKC17] with the exception that the spline is not confined to only have cable points placed on the nodes of the mesh.

In this chapter, we formulate the model we will be using to compute cable actuation forces directly - based primarily on the work of [Eng12] and [BCC] - and describe how the model can be implemented in a simulator.

4.1 Modelling Cable Forces

As in [BCC], we will define our cables as piece-wise linear segments with two end-points and $n \geq 0$ via points, but we allow for cable points to be placed anywhere on our mesh, as long as each point lies within an element of the mesh. This means that a cable is defined by the set of via points $\{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n\}$ and the attachment points $\mathbf{x}_0$ and $\mathbf{x}_{n+1}$, as can be seen in the example in Figure 4.1. Our cable has a direction, which we define as going from the point $\mathbf{x}_0$ to the point $\mathbf{x}_{n+1}$; that is, from the starting point to the end point.

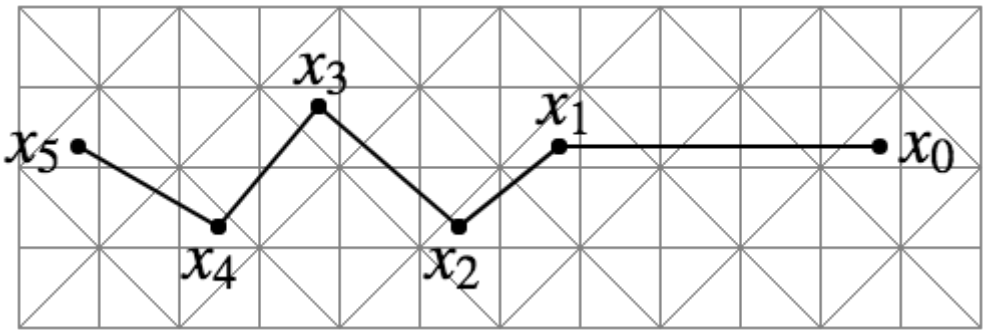


Figure 4.1: 2D-representation of a robot with a cable embedded consisting of via points $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$ and attachment points $\mathbf{x}_0$ and $\mathbf{x}_5$.

For a cable consisting of $n+2$ points, we define the $n+1$ segments of the cable as,

$$\mathbf{l}_i = \mathbf{x}_{i+1} - \mathbf{x}_i \ , \quad \forall i \in \{0, 1, \cdots, n\}$$

corresponding to a unilateral spring from the point $\mathbf{x}_i$ to the point $\mathbf{x}_{i+1}$. For each of these $n+1$ segments, we define the length of each segment as,

$$l_i = \|\mathbf{l}_i\|_2 \ , \quad \forall i \in \{0, 1, \cdots, n\}$$

and the direction of each segment as,

$$\hat{l}_i = \frac{\mathbf{l}_i}{l_i} \ , \quad \forall i \in \{0, 1, \cdots, n\}$$

Now, as our cable segments are modelled as unilateral springs, we know from [Erl+05] that the force magnitude at each segment of the cable is given by,

$$f_i^{(t)} = -k(l_i^{(t)} - l_i^{(0)}) \ , \quad \forall i \in \{0, 1, \cdots, n\} \tag{4.1}$$

if we ignore the damping of the spring. In other words, the force magnitude $f_i^{(t)}$ is defined by the spring stiffness, k, and the difference between the current length and the rest length of the cable segment.

To model pulling of a cable, we will proceed as in [Eng12] and [BCC] by introducing a control parameter $\alpha^{(t)}$, which we can use to change the rest length of the cable, and thus, create a force,

$$f_i^{(t)} = -k\Big(l_i^{(t)} - \alpha^{(t)} l_i^{(0)}\Big), \forall i \in \{0, 1, \cdots, n\} \tag{4.2}$$

This force is that of a spring, which means that it is able to exert forces both while contracting and extending. As we are modelling cables, we do not want to produce any forces when the cable is extended, as it should have no pushing power. In other words, we want to model the cable 'slack', and therefore, we only want to produce a force when the controlled rest length of the cable is smaller than the current length. This means that the force magnitude of any cable segment is instead given by,

$$f_i^{(t)} = -k \max\Big(l_i^{(t)} - \alpha^{(t)} l_i^{(0)}, 0\Big), \forall i \in \{0, 1, \cdots, n\} \tag{4.3}$$

and since the cable forces work in the direction of the cable segment, we get a directional force at each segment of,

$$\mathbf{f}_i^{(t)} = f_i^{(t)} \cdot \hat{l}_i \ , \quad \forall i \in \{0, 1, \cdots, n\} \tag{4.4}$$

Now, as our cables are modelled as springs, we need to add a dampening term in order to avoid the cable oscillating. As in [Eng12] and [Erl+05], we will add damping at every cable segment, which means that for the segment from $\mathbf{x}_i$ to $\mathbf{x}_{i+1}$ we have a spring force on $\mathbf{x}_{i+1}$ of,

$$\mathbf{a}_{i+1}^{(t)} = \mathbf{f}_i^t - b \cdot \hat{l}_i^{(t)} (\hat{l}_i^{(t)})^T (\mathbf{v}_{i+1}^{(t)} - \mathbf{v}_i^{(t)}) \tag{4.5}$$

where b is a damping coefficient and $\mathbf{v}_i^{(t)}$ is the velocity of the i'th cable point. We approximate the velocities of our cable points as the weighted velocity of the mesh nodes of the element in which each cable point is embedded. To get the real cable point velocities, we would need to explicitly simulate the cable, as the velocity of the points depend on quantities such as friction between the cable and robot body, and the mass of the cable. From Newton's third law of motion, we have a force of,

$$\mathbf{a}_i^{(t)} = -\mathbf{a}_{i+1}^{(t)}$$

at the i' th cable point. For each via point on the cable, we experience a cable force, $\mathbf{F}_i^{(t)}$, given by,

$$\mathbf{F}_i^{(t)} = \mathbf{a}_i^{(t)} - \mathbf{a}_{i+1}^{(t)} \tag{4.6}$$

and for attachment points $\mathbf{x}_0$ and $\mathbf{x}_n + 1$ a force of,

$$\mathbf{F}_0^{(t)} = -\mathbf{a}_1^{(t)}$$
$$\mathbf{F}_{n+1}^{(t)} = \mathbf{a}_n^{(t)}$$

We discretise our robots as tetrahedral meshes, and we embed cables into the mesh by binding each point to a tetrahedron. This means that the positions and velocities of the attachment and

via points of the cable changes over time. As we only know the real world coordinates of the cable points at the beginning of the simulation, we need to re-interpolate both positions and velocities at the beginning of each simulation step. At the beginning of the simulation, we therefore compute the barycentric coordinates of each point in terms of the tetrahedron it is bound to. In Figure 4.2, a point $\mathbf{x}_p$ can be seen embedded into the tetrahedron consisting of nodes i, j, k and m.

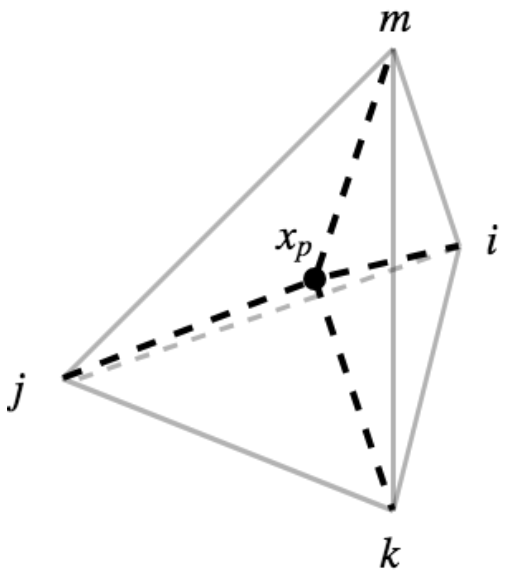


Figure 4.2: A point $\mathbf{x}_p$ embedded into the tetrahedral element with vertices i, j, k, m, with sub-tetrahedra marked by the dashed lines.

The barycentric coordinates of the point $\mathbf{x}_p$ bound to the tetrahedron with vertices i, j, k and m, are given by

$$\mathbf{x}_p = w_i \cdot i + w_j \cdot j + w_k \cdot k + w_m \cdot m$$

where

$$w_i = \frac{V_{j,k,m,\mathbf{x}_p}}{V_{i,j,k,m}}, \quad w_j = \frac{V_{i,k,m,\mathbf{x}_p}}{V_{i,j,k,m}}, \quad w_k = \frac{V_{m,i,j,\mathbf{x}_p}}{V_{i,j,k,m}}, \quad w_m = \frac{V_{i,j,k,\mathbf{x}_p}}{V_{i,j,k,m}}$$

and $V_{1,2,3,4}$ is the volume of the tetrahedron consisting of vertices $1, 2, 3$ and 4. This is also the weighting used to compute the velocities of our cable points.

We can now couple the cable forces, $\mathbf{F}^{(t)}$, to the nodal forces we want to apply to the mesh, $\mathbf{N}^{(t)}$, as

$$\begin{aligned} \mathbf{F}_0^{(t)} &= w_i \cdot \mathbf{N}_i^{(t)} + w_j \cdot \mathbf{N}_j^{(t)} + w_k \cdot \mathbf{N}_k^{(t)} + w_m \cdot \mathbf{N}_m^{(t)} \\ &\vdots \\ \mathbf{F}_{n+1}^{(t)} &= \dots \end{aligned} \tag{4.7}$$

where $\mathbf{N}_i^{(t)}$ is the force at node i of the tetrahedron to which the point is bound. For a cable with M points, bound to a tetrahedral mesh with a total of N nodes, we can construct the $(M \times N)$ weight matrix $\mathbf{W}$ for which

$$\mathbf{W}_{ij} = \begin{cases} w_j & \text{if } j \in T(i) \\ 0 & \text{otherwise} \end{cases}$$

where $T(i)$ is the set of vertices of the tetrahedron that contains point $\mathbf{x}_i$. Using this definition of the weight matrix we can express the force computations as,

$$\mathbf{F}^{(t)} = \mathbf{W}\mathbf{N}^{(t)} \tag{4.8}$$

Now, as we will almost always have less cable points than nodes in our mesh, $\mathbf{W}$ will most of the time have more columns than rows. Every cable point is bound to exactly one tetrahedron, which means that $\mathbf{W}$ will have full row rank most of the time. However, certain embeddings can result in a weight matrix with linearly dependent rows, but we will assume that this does not happen for our robots. Observe in Figure 4.3 an example - in 2D - of a configuration in which the weight matrix does not have full row rank.

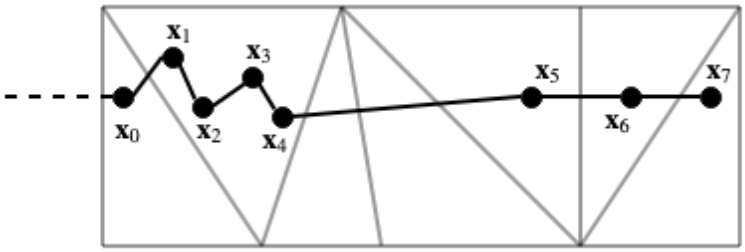


Figure 4.3: The cable points $\mathbf{x}_0, \mathbf{x}_1 \ldots, \mathbf{x}_7$ embedded in a mesh, where $\mathbf{W}$ would be rank deficient.

Here, the nodes $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$ and $\mathbf{x}_4$ are all embedded in the same triangle element. Thus, in the weight matrix $\mathbf{W}$ we would have four rows with values in exactly the same columns, which would mean that we no longer only have linearly independent rows in $\mathbf{W}$, and therefore, the matrix will not have full row rank.

We want to find the nodel forces, $\mathbf{N}^{(t)}$, to apply to our mesh at time (t) given the cable forces $\mathbf{F}^{(t)}$ and weight matrix $\mathbf{W}$. As $\mathbf{F}^{(t)} = \mathbf{W}\mathbf{N}^{(t)}$ is an underdetermined system, $\mathbf{W}$ will not be invertible. Fortunately, $\mathbf{W}$ has full row rank, which means we can use its pseudo-inverse, $\mathbf{W}^\dagger$, to find the nodal forces we wish to apply to the mesh. By substituting $\mathbf{N}^{(t)} = \mathbf{W}^T\mathbf{y}^{(t)}$, we get,

$$\begin{aligned}
\mathbf{W}\mathbf{N}^{(t)} &= \mathbf{F}^{(t)} \Rightarrow \\
\mathbf{W}\mathbf{W}^T y &= \mathbf{F}^{(t)} \Rightarrow \\
y &= (\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{F}^{(t)}
\end{aligned}$$

and inserting back into $\mathbf{N}^{(t)} = \mathbf{W}^T\mathbf{y}^{(t)}$ yields,

$$\begin{aligned}
\mathbf{N}^{(t)} &= \mathbf{W}^T\mathbf{y}^{(t)} \Rightarrow \\
\mathbf{N}^{(t)} &= \mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}\mathbf{F}^{(t)}
\end{aligned}$$

where $\mathbf{W}^T(\mathbf{W}\mathbf{W}^T)^{-1}$ is the Moore-Penrose pseudo-inverse, $\mathbf{W}^\dagger$. Hence, we can derive the nodal forces as,

$$\begin{aligned}
\mathbf{F}^{(t)} &= \mathbf{W}\mathbf{N}^{(t)} \Rightarrow \\
\mathbf{N}^{(t)} &= \mathbf{W}^\dagger\mathbf{F}^{(t)}
\end{aligned} \tag{4.9}$$

4.2 Implementation Design

The cable force model described in Section 4.1 computes the cable forces as external forces affecting the tetrahedral elements of the soft robot mesh. The barycentric coordinates and the initial rest

length can be computed at the beginning of the simulation, which means that we only need to know the current positions and velocities of the mesh nodes to compute the external forces introduced by a cable. This makes it easy to attach our cable model to an existing simulation framework, as it should be trivial to add external forces to the nodes of the mesh. Pseudo-code for the force computation that need to be executed at each simulation step can be found in Algorithm 1.

Algorithm 1: Computing Cable Forces At Cable Points

Data:

$\mathbf{W}$: Barycentric coordinates of the cable,
$\mathbf{W}^{\dagger}$: Pseudo-inverse of $\mathbf{W}$,
$\mathbf{v}^{(t)}$: Vertex-velocities of tetrahedral elements that contain via points,
$\mathbf{p}^{(t)}$: Vertex-positions of tetrahedral elements that contain via points,
$l^{(0)}$: The original rest length of the cable,
$\alpha^{(t)}$: The contraction of the cable at time t,
k: The stiffness coefficient of the cable,
b: The damping coefficient of the cable,

begin

/* Step 0: Re-interpolate cable position P, velocities V and compute cable geometry */

$$\mathbf{P}^{(t)} = \mathbf{W}\mathbf{p}^{(t)}$$
$$\mathbf{V}^{(t)} = \mathbf{W}\mathbf{v}^{(t)}$$
$$\mathbf{l}_i^{(t)} = \mathbf{P}_{i+1}^{(t)} - \mathbf{P}_i^{(t)}, \qquad \forall i \in \{0, 1, \cdots, n\}$$
$$l_i^{(t)} = \left\| \mathbf{l}_i^{(t)} \right\|, \qquad \forall i \in \{0, 1, \cdots, n\}$$
$$\hat{l}_i^{(t)} = \frac{\mathbf{l}_i^{(t)}}{l_i^{(t)}}, \qquad \forall i \in \{0, 1, \cdots, n\}$$
$$f_i^{(t)} = -k \max\left(l_i^{(t)} - \alpha^{(t)} l_i^{(0)}, 0\right), \qquad \forall i \in \{0, 1, \cdots, n\}$$
$$\mathbf{f}_i^{(t)} = f_i^{(t)} \cdot \hat{l}_i, \qquad \forall i \in \{0, 1, \cdots, n\}$$

/* Step 1: Compute forces at each cable point */

$$\mathbf{a}_{i+1}^{(t)} = \mathbf{f}_i^{(t)} - b \cdot \hat{l}_i^{(t)} (\hat{l}_i^{(t)})^T (\mathbf{V}_{i+1}^{(t)} - \mathbf{V}_i^{(t)}), \qquad \forall i \in \{0, 1, \cdots, n\}$$
$$\mathbf{F}_i^{(t)} = \mathbf{a}_i^{(t)} - \mathbf{a}_{i+1}^{(t)}, \qquad \forall i \in \{1, \cdots, n\}$$
$$\mathbf{F}_0^{(t)} = -\mathbf{a}_1^{(t)}$$
$$\mathbf{F}_{n+1}^{(t)} = \mathbf{a}_{n+1}^{(t)}$$

/* Step 3: Couple cable forces to the mesh */

$$\mathbf{N}^{(t)} = \mathbf{W}^{\dagger}\mathbf{F}^{(t)}$$

return $\mathbf{N}^{(t)}$

end

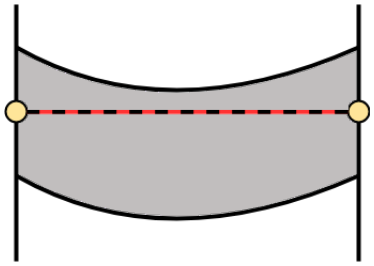
The cable model allows for the addition of multiple cables in a single mesh, and since the cable forces can be computed independently it would be easy to parallelise the force computations.

Adding cable forces as external forces makes the coupling between the simulator and our cable model easy to implement. However, to accurately depict the real world in our simulations we need the stiffness, k, to be very large. Assuming that the simulator performs an implicit or explicit time integration to compute elastic forces of the system, the size of the steps taken by the solver needs to be very small, as we would otherwise approximate any time derivatives poorly. Our model depends on the assumption that each cable is made of almost infinitely stiff material, which means

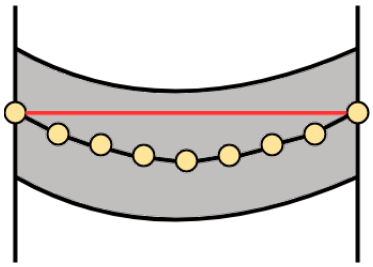
that we need a simulator capable of swiftly computing the elastic forces of the system. To perform all of our simulations we will be using the NVIDIA Flex simulator - more specifically, NVIDIA Flex (Robotics Alpha) 2.0.0 - due to its fast deformable object simulations.

4.2.1 Discretisation Considerations

To make sure that we can accurately estimate the length of the cable at any time t, we need to discretise it in such a way that the cable curving can be adequately approximated. If we use too few points, the length that we measure will not be accurate and we will produce unrealistic simulations. Observe in Figure 4.4 two identical robots with a different amount of via points approximating a cable going from end to end of a soft body which is fixed at both ends. In this example, it is clear that the low resolution cable will not be adequate, as the length of the cable will never change.



(a) A low resolution cable approximation of a cable going through a soft robot.



(b) A higher resolution cable approximation of a cable going through a soft robot.

Figure 4.4: Two approximations of a cable at different resolutions. Note that the red line is the cable at time $t = 0$, while the black line is the cable approximation at the current time step and the yellow nodes are the via points of the cable. Observe that in the low resolution case, the black line is identical to the red line, which means that the low resolution is not suitable for creating realistic simulations in this case.

In [Ber+19], placing approximately one cable point in each tetrahedron the cable passes through seems to have shown realistic deformations, but this topic needs to be further investigated to provide more than a rule-of-thumb in terms of how the cable should be discretised.

The discretisation of the cable is only one of the parameters of the simulation that influence the physical accuracy. As we want to create simulations of soft robots being actuated, we need to make sure that we know correct material properties, such as the Young's Modulus and Poisson's ratio. Finally, we also need to determine our cable parameters, k and b, as well as how fast the cable can be pulled, or in other words, how quickly $\alpha^{(t)}$ can change. For changes in the rest length we want to avoid creating an excessive amount of elastic forces, but we still need to make sure that the forces are large enough to reflect the behaviour of the silicone in a real world environment. To ensure realistic simulations, we examine the speed of the physical motors we will be using in Experiment 4.3.1, as this provides a baseline for how quickly $\alpha^{(t)}$ should be allowed to change.

In terms of choosing stiffness and damping parameters, we ideally want critically damped or over damped springs. We could proceed as in [Erl+05], by choosing our parameters as,

$$k = \frac{1}{\tau^2}, \text{ and } b = \frac{2}{\tau} \tag{4.10}$$

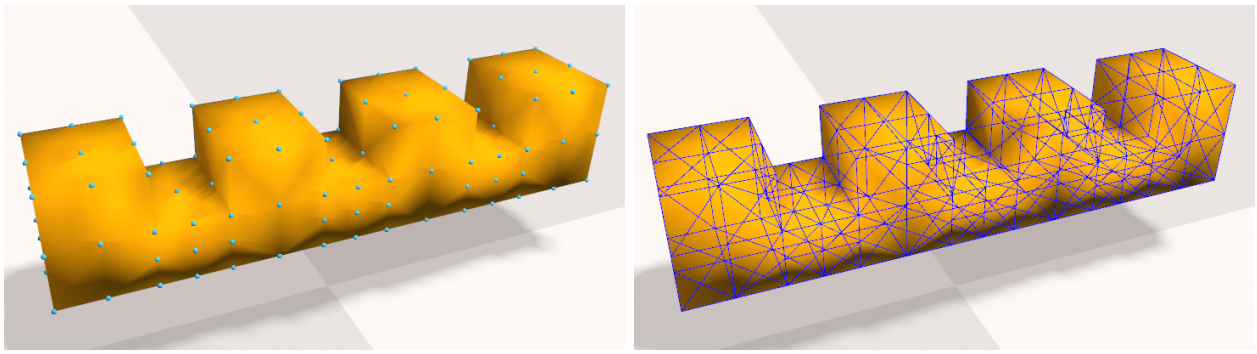
where

$$\tau = \frac{\Delta t}{\lceil -\varepsilon \rceil}$$

and Δt is the time step size of the simulator, and $0 < \varepsilon << 1$, to ensure that the springs are critically damped. However, when the time step Δt is large we would see enormous spring forces which in turn creates large errors in the simulation. In the NVIDIA Flex simulator we are very limited in terms of how low Δt can be set, and therefore, we will not choose parameters as in Equation (4.10). We will however try to maintain the same relationship between the parameters as described above, using values within an order of magnitude of 10^3 and 10^{-3} for k and b, respectively.

4.2.2 Implementing the Cable Model in NVIDIA Flex

The traditional approach to simulate dynamic objects has long been to work with forces. This means that at the beginning of each simulation step internal and external forces are collected and translated into accelerations following Newton's second law of motion. The accelerations are then integrated by some time integration scheme to compute velocities, which in turn are integrated to derive positional updates. For this thesis we will be using the GPU accelerated NVIDIA Flex simulator [Cor19b], due to the speed and stability it provides during simulations of deformable objects. Flex implements the Newton method described in [Mac+19], which is a position based dynamics solver that further develops on the XPBD algorithm [MMC16], instead of the traditional simulation loop. This type of simulator controls the movement of the mesh by placing a number of *particles* on the surface, for which a series of *constraints* are added. Each of these constraints is responsible for ensuring that the mesh behaves properly in terms of colliding with other objects, avoiding self-intersections, etc. Observe in Figure 4.5 an example of a robot with particles visualised as cyan spheres and the triangulated surface visualised in blue.



(a) Particles on an object.

(b) The triangulation of the surface.

Figure 4.5: An example of a soft deformable object. Notice that the particles on the surface mesh match the vertices of the triangulation.

At the beginning of each simulation step the new positions of the mesh particles are predicted based on their current velocities and the potential energy of the system. As these positions are subject to a number of constraints the predicted positions need to be updated such that all of the constraints are respected before the simulation can continue to the next iteration. From [MMC16], the prediction of the new particle positions are given by

$$\mathbf{x}^n \leftarrow \mathbf{x}^{n-1} + \Delta t \mathbf{v}^{n-1} + \Delta t^2 \mathbf{M}^{-1} \mathbf{f}_{ext}(\mathbf{x}^{n-1}) \tag{4.11}$$

where $\mathbf{x}^n$ is the predicted position, $\mathbf{v}^{n-1}$ is the velocity in the n'th iteration, $\mathbf{M}^{-1}$ is the inverse mass matrix, and $\mathbf{f}_{ext}(\mathbf{x}^{n-1})$ are the external forces on the nodes of the mesh. Now, our cable

model computes external forces at each node of the mesh, and ideally we would want to add our external forces to the initial prediction, or more specifically, to $\mathbf{f}_{ext}(\mathbf{x}^n)$. Unfortunately, due to the limited exposure of the internal simulation loop of the Flex simulator, we will add our forces using an explicit time integration, such that at each iteration - before constraints are resolved - the simulator will perform the following update:

$$\mathbf{v}^{\sim} \leftarrow \mathbf{v}^{n-1} + \mathbf{w} \cdot \mathbf{N}^{n-1} \cdot \Delta t \tag{4.12}$$

$$\mathbf{x}^{\sim} \leftarrow \mathbf{x}^{n-1} + \mathbf{v}^{\sim}\Delta t \tag{4.13}$$

$$\mathbf{x}^{n} \leftarrow \mathbf{x}^{\sim} + \Delta t\mathbf{v}^{\sim} + \Delta t^2\mathbf{M}^{-1}\mathbf{f}_{ext}(\mathbf{x}^{\sim}) \tag{4.14}$$

where $\mathbf{v}$ is velocity, $\mathbf{w}$ is the inverse mass, and $\mathbf{N}$ is the force found by the cable model. In future work the cable forces should be implemented as part of the solver, but for now we will be using the explicit approach described above and ignore any instability that it introduces.

The cable model from Algorithm 1 assumes that each cable point is embedded inside of a tetrahedron, but in Flex we can only interact with the surface of the dynamic objects directly. In terms of the use of barycentric coordinates for constructing the weight matrix, $\mathbf{W}$, this is of little consequence, as the cable point lying in the plane of a triangle of a tetrahedron is just a corner case of the embedding procedure. This is illustrated in Figure 4.6, where the cable point $\mathbf{x}_p$ is placed in the plane of triangle (i, j, k) of tetrahedron (i, j, k, m).

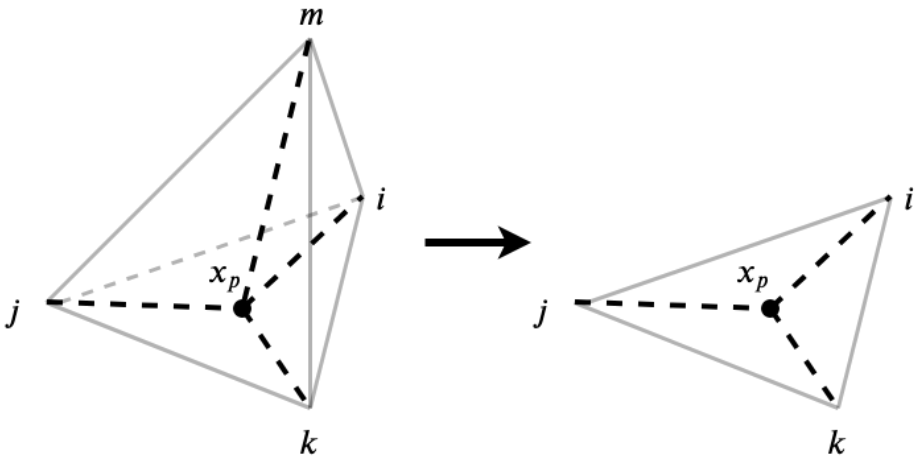


Figure 4.6: A cable point $\mathbf{x}_p$ is placed within the plane of one of the triangles of the tetrahedron with vertices i, j, k and m. Here, the barycentric coordinates would be the same if the cable point was directly embedded in the triangle with vertices i, j and k, as $\mathbf{w}_m$ would be 0.

On the other hand, this limits the fidelity of our cable model, as we can no longer discretise the cable according to our rule-of-thumb, and we might therefore see problems such as the one showcased in Figure 4.4. Observe in Figure 4.7 an example of how a cable can be placed on a robot.

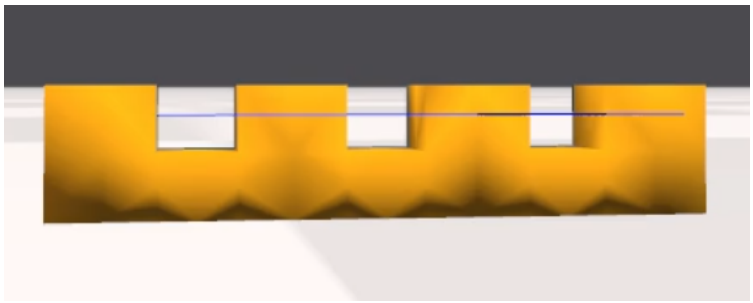
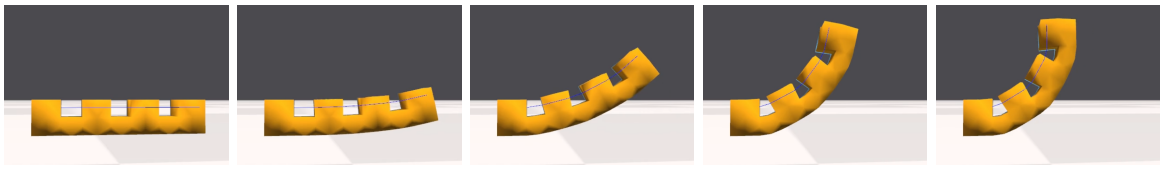


Figure 4.7: An example of how a cable can be placed on a robot. The cable is visualised in blue, and passes through cable points on each side of the wedges of the robot, except the first, where it is only connected on one side.

Using the cable model and the time integration scheme described above, we have been able to create simulations such as the ones shown in the still images that can be seen in Figure 4.8. Please refer to the web page `mystiking.github.io` for videos of the robot during actuation.



(a) (b) (c) (d) (e)

Figure 4.8: An example of a soft robot being deformed by a cable being pulled. Here the particles of the left-most joint of the robot has an inverse mass of 0, which makes them immovable. The cable is visualised as a series of blue connected segments.

4.3 Experiments

4.3.1 Motor Speed

Introduction

Our cable model is dependent on the speed at which the rest length of a cable is able to change. To produce realistic simulations we need to know the velocity at which our stepper motors can reel in a cable, as this allows us to choose realistic $\alpha^{(t)}$ values during simulation.

Procedure

We will examine the speed at which a stepper motor is able to reel in a cable by measuring the time it takes to do 10 mm, 20mm, 30 mm, ..., 200 mm worth of revolutions. The motors used are the Mercury 2-phase motors [Ele19c], and the time measured will be the time it takes to execute a `setPos` command from the `pySoRo` [HE18] motor control API. The set-up can be seen in Figure 4.9.

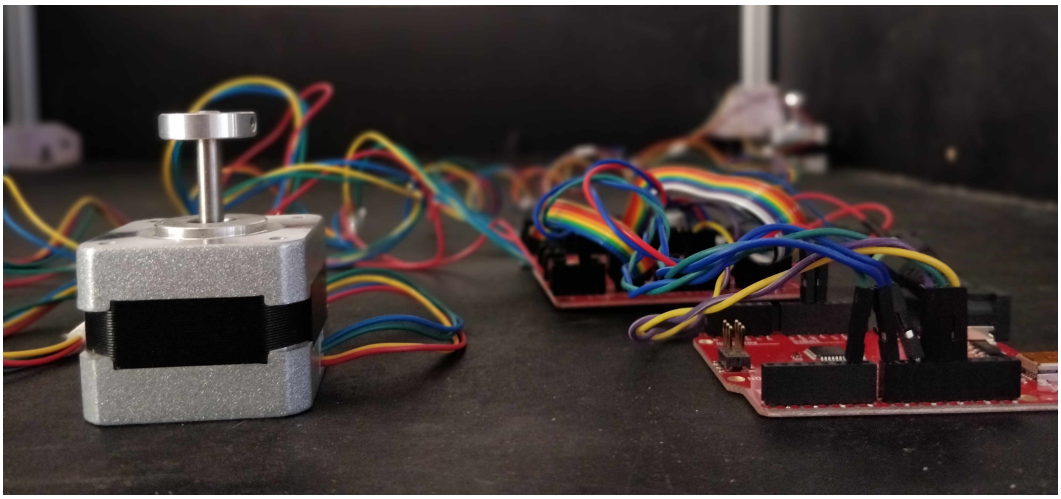


Figure 4.9: The set-up for the motor speed experiment using a single motor.

We run the experiment 10 times for each length to lower the variance of the experiment and present both the mean result and standard deviation in terms of milliseconds needed to complete each command.

Results

Observe in Figure 4.10 the mean times recorded along with the standard deviation of the time over the 10 runs.

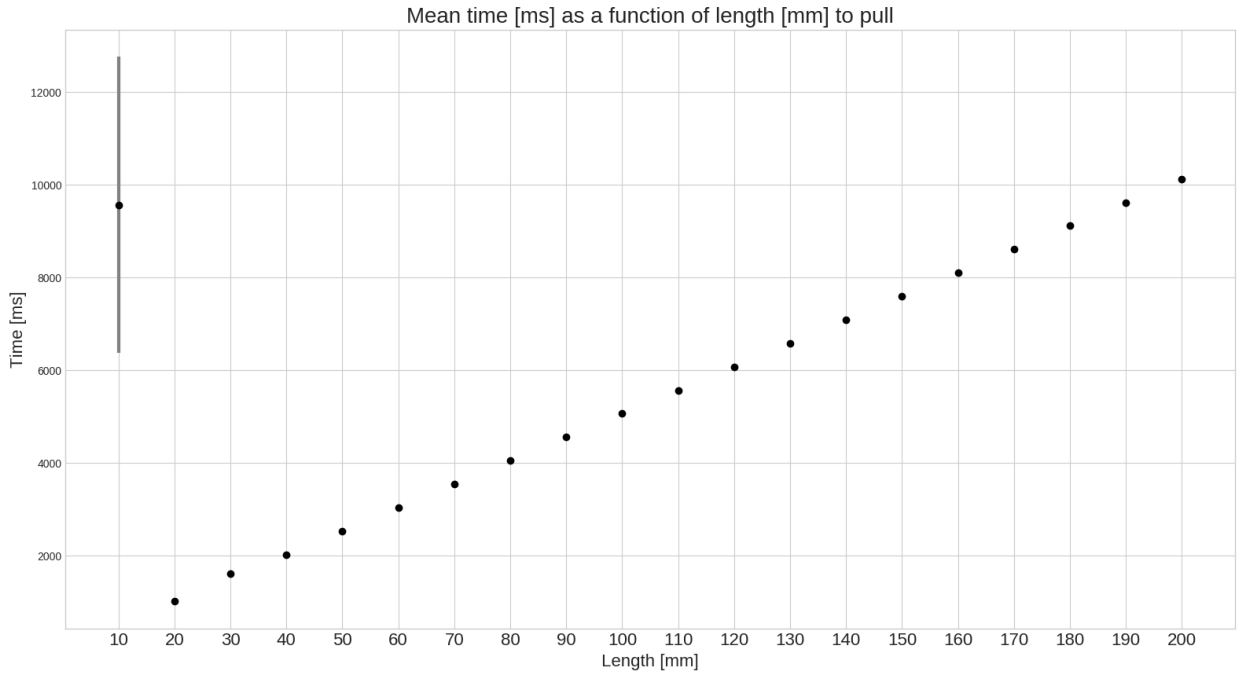


Figure 4.10: Mean time in milliseconds it took to pull a cable 10 mm, 20 mm, ..., 200 mm, plotted with standard deviation. The standard deviation is so low for every length except the first that it cannot be discerned in the plot.

As expected, the time taken increases linearly as the length increases for all pulling lengths

from 20 mm to 200 mm. However, for the 10 mm pull the time taken was notably larger. The experiment was performed such that the sequence of pulls were 10 mm, 20mm, ... 200 mm, 10 mm, ..., 200 mm, ..., and it seems that after having taken 200 mm worth of steps, the motor was not able to accurately pull 10 mm. This indicates that the `Mercury` 2-phase stepper motors might be inaccurate in terms of performing a large amount of revolutions followed by a small amount of steps, which we will take into account during future experiments.

In terms of ensuring realistic simulations it seems reasonable to pick $\alpha^{(t)}$ by using the first order function that approximates the data from Figure 4.10 - except the 10 mm data point. In Figure 4.11, the 20 mm to 200 mm data points can be found with fitted first order function $f(x)$ in blue.

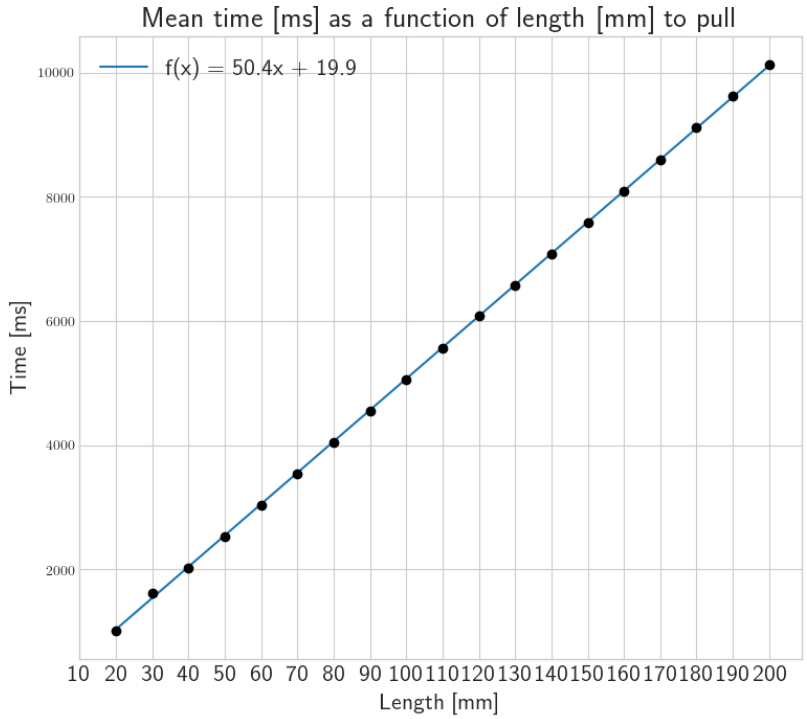


Figure 4.11: Mean time in milliseconds it took to pull a cable 20 mm, 30 mm, ..., 200 mm, and the first order function $f(x)$ that approximates the data points.

Thus, for future experiments we will use the function $f(x) = 50.4x + 19.9$ to determine how much time it should take to pull a cable x mm.

4.3.2 Choosing a Suitable Stiffness Coefficient

Introduction

As described in Section 4.2.1, we cannot choose a small enough Δt to make sure that the underlying spring model of our cable discretisation is not under damped. However, as the elastic forces of the silicone body of our soft robots also has a dampening effect on the embedded cables, we might still be able to find a stiffness coefficient for which our cable oscillates minimally.

Procedure

For this experiment we will actuate a simple cable-driven robot, which is fastened to an invisible wall in one end. The robot is hanging upside down, and is subject to gravitational forces. Observe in Figure 4.12 the volume mesh of the robot that we will be using in this experiment.

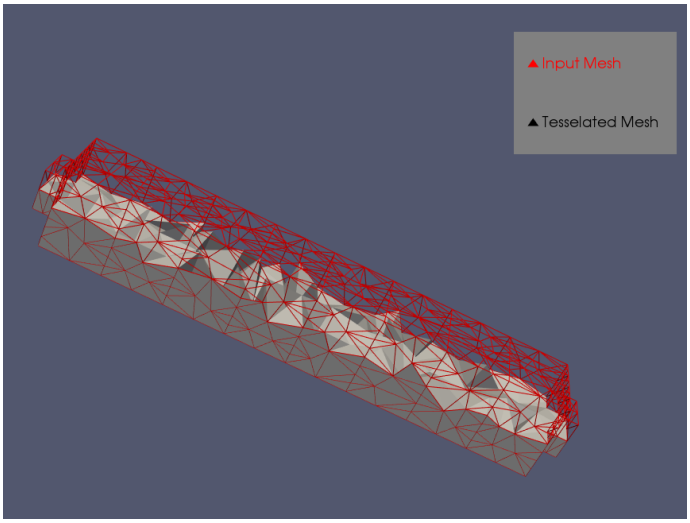


Figure 4.12: The tetrahedral mesh of the robot that we will actuate in this experiment.

A single cable is added to the robot - stretching from end to end - measuring exactly 130 mm. During the simulation, the resting length of the cable will be changed to 95% of its original length, which means that at the end of the simulation we expect to measure the cable length at 123.5 mm. Using the results from the previous experiment, a reduction of the cable length by 6.5 mm should take 347.5 ms, but since we want to be able to observe any oscillations in the cable, the experiment will be allowed to run for a total of 800 ms. Observe in Figure 4.13 the set-up of the experiment inside the Flex simulator at the beginning of the simulation.

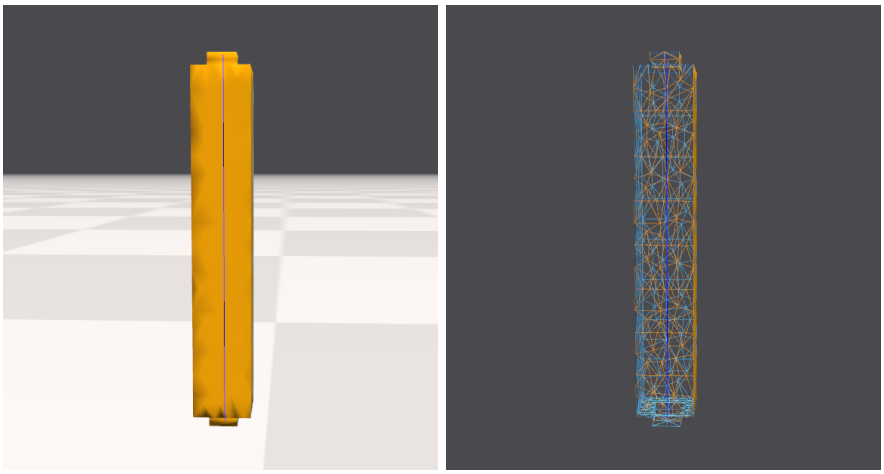


Figure 4.13: The robot instantiated in Flex, next to its wireframe. The robot is fastened in the top, which means that it will hang in place during the simulation.

To emulate our material of choice for soft robot fabrication - Ecoflex 00-50 - we will use the

material properties from Table 4.1, which are based on the work of [Ati12; Smo; Gas+19].

Young's modulus (E)	0.2642 MPa
Poisson's ratio (ν)	0.4999
Density (ρ)	1.07 gram / cc

Table 4.1: Values used to approximate the behaviour of Ecoflex 00-50 in the simulations.

We will perform the experiment for all the stiffness coefficients given in Table 4.2, but for all of our simulations we will use a damping coefficient of 0.001.

Experiment	**Stiffness**
Experiment 4.3.2.a	1000.0
Experiment 4.3.2.b	2000.0
Experiment 4.3.2.c	3000.0
Experiment 4.3.2.d	4000.0
Experiment 4.3.2.e	5000.0
Experiment 4.3.2.f	10000.0
Experiment 4.3.2.g	25000.0

Table 4.2: The stiffness coefficients we will use in this experiment.

Results

Observe in Figure 4.14 the change in cable length as a function of time for all of the stiffness coefficients from Table 4.2.

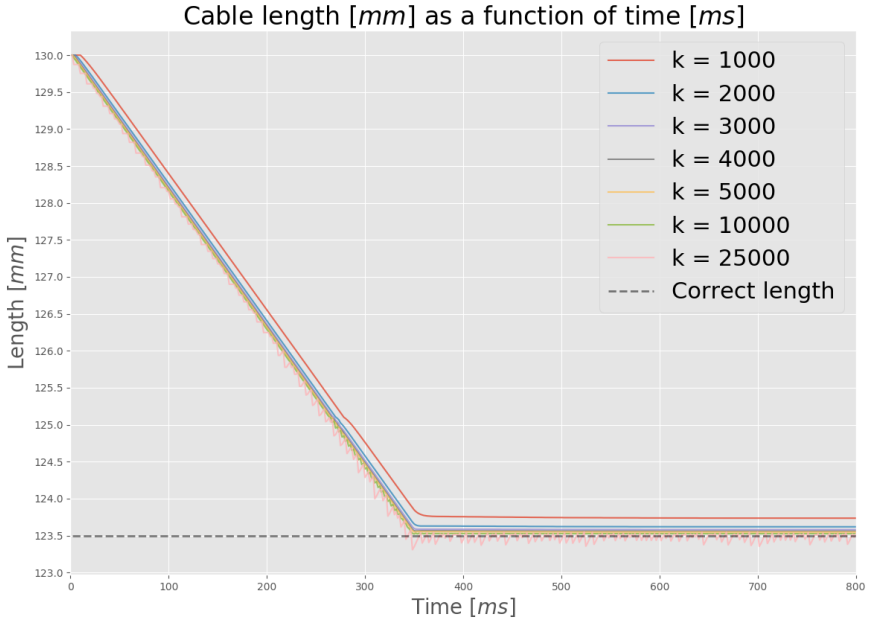
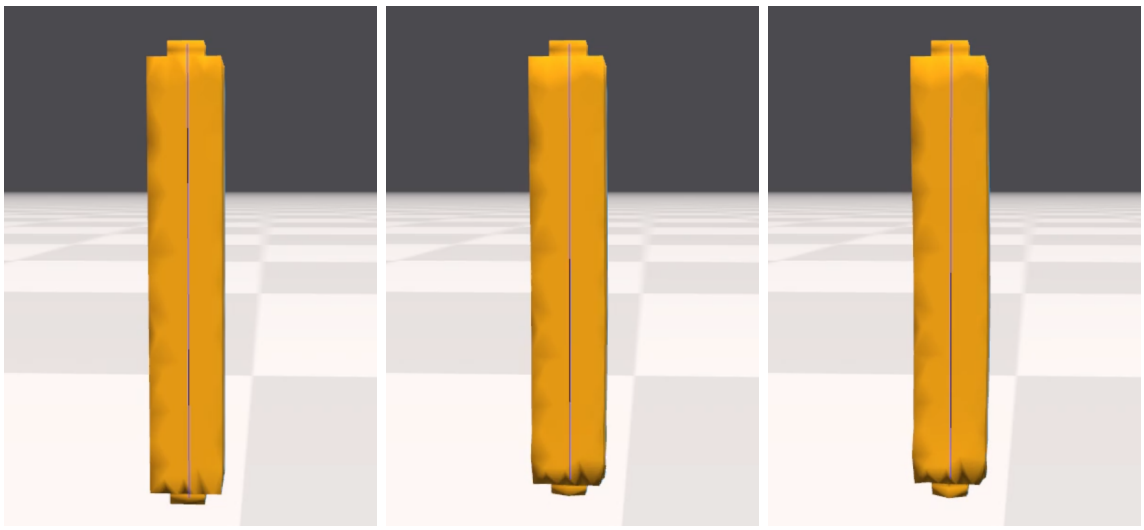


Figure 4.14: The changes to cable length over time, for different stiffness coefficients. The dotted line corresponds to the expected cable length at the end of the simulation.

Notice that for stiffness coefficients $2000, \cdots, 5000$ we observe the behaviour that we expect in real cables - no oscillations during the reduction of the rest length and a cable length at the end of the simulation at approximately 123.5 mm at a $\sim 1-2$ mm precision. However, we also observe that for large stiffness coefficients we actually do see oscillations in cable length, which indicates that the time step size is too large for these stiffness values. In Figure 4.15 still shots of the robot during simulation can be found.



(a) . (b) . (c) .

Figure 4.15: The robot actuated using stiffness coefficient $k = 3000$. As the cable is only pulled 6.5 mm, the deformations in the soft body are minimal.

As expected, a rather small deformation is created, as the cable is only pulled 6.5 mm, but it is important to note that we do not see any visually unpleasant deformations. Thus, in future experiments we will use a stiffness value of $k \in \{2000, \cdots, 5000\}$, with the exact value determined for each experiment based on a visual validation.

Chapter 5

Pneumatic Actuation

In recent soft robotics papers, pneumatically actuated robot designs have proven to be especially well suited for grasping objects of varying geometries. In [Gli+18] the combination of an adhesive layer and an asymmetric air-chamber design aimed to maximise the area of contact between the actuator and the target surface. The structural asymmetry in the robot was used to convert the expansion of the air-chambers into a bending motion, and distribute the load uniformly across the adhesives. To model and control the soft gripper, the authors constructed a torque model allowing for the design of the soft robot to meet a specific torque profile. In short, they related the internal pressure and chamber geometry directly to the torque at each joint - or wedge, using our terminology - as,

$$\tau_{\text{joint}} = P \cdot A_f \cdot d$$

where P is the internal pressure, A_f is the area of the face in contact - i.e. the area of the inner surface of the air-chamber - and d is the distance from the neutral axis of the robot to the centre of the air-chamber.

In [Hao+17] a similar approach was used to model a soft robot subject to pressure forces. The authors also developed a soft gripper, and controlled the deformations of the robot by using a model based on air-chamber geometry. However, instead of modelling torque, the deformations of the robot was determined based on the stress that a pressure force would place on the walls of the chamber.

More recently, in [Gas+19], actuation was performed by inducing a strain, causing an expansion or compression in the air-chamber of the soft robot. Now, unlike the previous approaches, this scheme was implemented in the Flex simulator by modelling the pressure forces as constraints between particles. Given a pressure p, and a rest length r, two particles $\mathbf{q}_i$ and $\mathbf{q}_j$ of an air-chamber were constrained such that,

$$c_{dist}(\mathbf{q}_i, \mathbf{q}_j, p) = |\mathbf{q}_i - \mathbf{q}_j| - r\varepsilon(p) = 0 \tag{5.1}$$

where $\varepsilon(p)$ is the strain induced by p on the soft body of the robot. In other words, pressure forces were modelled using an underlying spring model, similar to how cables were modelled in Chapter 4. Now, as it is not within the scope of this thesis to alter the Flex solver, we are limited to implementing pressure forces using an explicit time integration, as we did for the cable forces. However, none of the schemes described above have a clear way to be turned into an explicit positional update. We will therefore look to basic physics in order to compute pressure forces, as in [MO03] and [Erl+05] [1].

[1] In [Erl+05, Ch. 8.4.4], the pressure force computation is wrong, as it should be $\mathbf{F} = \mathbf{n}PA$ and not $\mathbf{F} = \mathbf{n}(P/A)$.

5.1 Modelling Pressure Forces

A pressure-driven soft robot is always subject to two antagonistic forces: the external forces trying to force an object to compress, and the internal forces making the object expand. If these two forces are not at an equilibrium, then the object will either expand or compress, depending on the magnitude of the opposing forces. From physics we know that pressure is force over area, which means that a surface with area A, affected by internal pressure P, will experience a force with magnitude,

$$\|\mathbf{F}\|_2 = PA \tag{5.2}$$

and, since the pressure is internal, the directional force affecting the surface is given by

$$\mathbf{F} = \mathbf{n}\,\|\mathbf{F}\|_2 = \mathbf{n}PA \tag{5.3}$$

where $\mathbf{n}$ is the normal of the surface. Observe in Figure 5.1 a hollow object that could be affected by internal pressure forces.

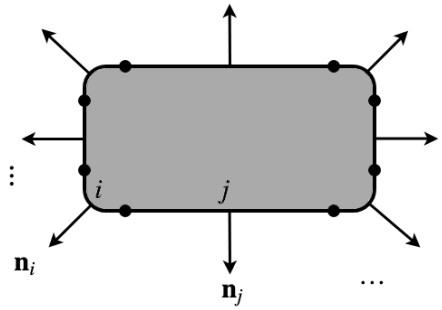


Figure 5.1: A sketch of an object and the directions - i.e. the normals - in which it would be subject to internal pressure forces.

The objects that we are interested in are soft silicone bodies, which means that instead of modelling pressure forces as internal forces, we will model them as an external force on the walls of the air-chambers of our robots. Observe in Figure 5.2 a sketch of the normals of a robot containing an air-chamber.

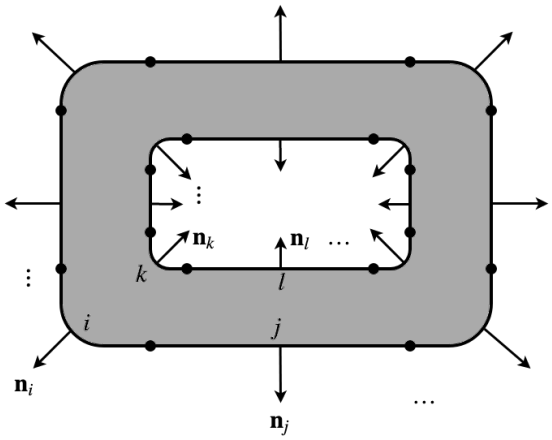


Figure 5.2: An object containing an air-chamber.

The external forces that should be applied to the walls of the air-chambers are given by

$$\mathbf{F} = -\mathbf{n}PA \tag{5.4}$$

in contrast to the internal forces. Pressure forces will always affect the entire surface of the air-chamber uniformly, which means that we can distribute forces across the finite elements of an air-chamber surface as,

$$\mathbf{F}_i = -\mathbf{n}_i PA_i \tag{5.5}$$

for the i'th element with area A_i and normal $\mathbf{n}_i$. During a simulation we want to be able to change the pressure inside the air-chambers, as this is how we will be controlling our pneumatically actuated robot. We will do this by using the *ideal gas law* to approximate changes to the pressure during simulation as described in [Erl+05] and [MO03]. The ideal gas law states that,

$$PV = nRT \tag{5.6}$$

where P is pressure, V is the volume of the object, n is the number of moles of gas, and R and T are the ideal gas constant and temperature, respectively. This means that we can determine the correct value of the pressure as,

$$P = \frac{nRT}{V} \tag{5.7}$$

but for practical purposes we usually choose a constant c as,

$$c = P^{(0)}V^{(0)} \tag{5.8}$$

such that we can compute the approximate pressure at time t as

$$P^{(t)} = \frac{c}{V^{(t)}} \tag{5.9}$$

The actuation devices that we will be using change the volume of the robot, through the use of a pump-like component, to increase or decrease the pressure. As the pump-like component will be made of a non-elastic material, we know that only the soft body of the robot can deform, and to ensure realistic simulations we can use the ideal gas approximation scheme to determine the new pressure value as we change the volume using the pump. For a pneumatically actuated soft body, we therefore introduce the control parameter $\beta^{(t)}$, which we use to determine the pressure current inside the robot, $P^{(t)}$, as,

$$P^{(t)} = \beta^{(t)}\frac{c}{V^{(t)}} \tag{5.10}$$

Now, to obtain an equilibrium in the external and internal forces affecting an air-chamber, we assume that at time $t = 0$, when $\beta^{(t)} = 1$, we have a force of exactly the same magnitude acting on the inside and outside of our robot. To include this consideration, we re-define $P^{(t)}$ as,

$$P^{(t)} = \beta^{(t)}\frac{c}{V^{(t)}} - P^{(0)} \tag{5.11}$$

since we will then have $P^{(t)} = 0$ when $\beta^{(t)} = \frac{V^{(t)}}{V^0}$ - i.e. when the current volume has been increased or decreased by a factor of $\beta^{(t)}$ - because,

$$\begin{aligned} P^{(t)} &= \beta^{(t)}\frac{c}{V^{(t)}} - P^{(0)} \Rightarrow \\ P^{(t)} &= \frac{V^{(t)}}{V^0}\frac{V^{(0)}P^{(0)}}{V^{(t)}} - P^{(0)} \Rightarrow \\ P^{(t)} &= P^{(0)} - P^{(0)} = 0 \end{aligned} \tag{5.12}$$

However, in this model we only approximate the outside pressure affecting our robot, as we model it by 'lowering' the pressure on the surface of our air-chambers, but in the real world, the external forces would affect all faces of the external surface of the robot.

5.2 Implementation Design

Our pressure force model computes forces at each surface - i.e. each face - but in our simulations we need to distribute those forces to the particles placed on the mesh. As we assumed that pressure is uniformly distributed, we need to distribute the forces evenly between the vertices of the elements of the air-chamber surface. Thus, the j'th vertex of the i'th face receive forces,

$$\mathbf{f}_j = \sum_{i \in T(j)} -\mathbf{n}_i \frac{P A_i}{C_i} \tag{5.13}$$

where $\mathbf{n}_i$ is the normal of the i'th face, C_i is the number of vertices making up the i'th face, A_i is the area of the i'th face, and $T(j)$ are the indices of the elements that node j is a part of. As the vertices can be shared between surfaces, they can also be affected by forces from multiple elements of the mesh, and therefore, the j'th vertex receives a a force from each of the air-chamber elements it is part of. The pseudo-code for the pressure model can be found in Algorithm 2, which computes the forces to apply at each vertex of a mesh.

Algorithm 2: Computing Pressure Forces

Data:

V: Current volume of the air-chamber
$P^{(0)}$: The initial pressure inside the air-chamber
$V^{(0)}$: The initial volume inside the air-chamber
$\mathcal{F}$: Set of faces of the surface of the air-chamber
$\beta^{(t)}$: The pressure control parameter

begin

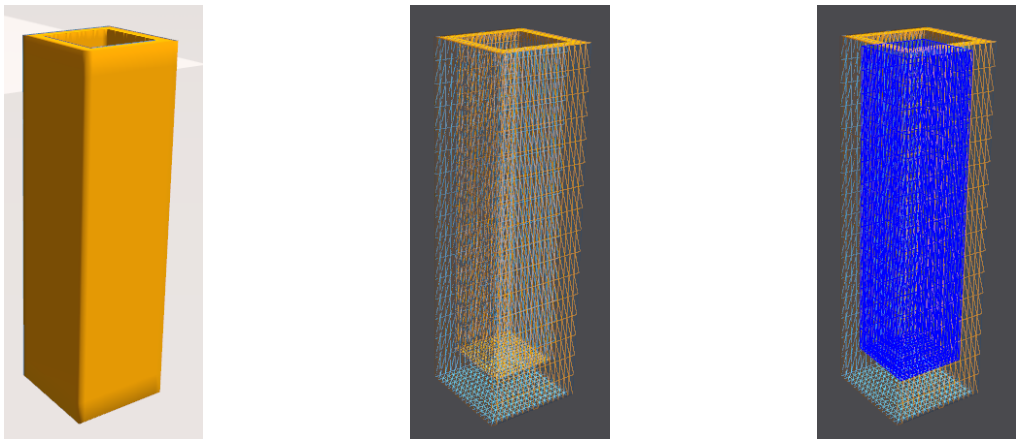
/* Step 0: Compute current pressure */
$c = P^{(0)} V^{(0)}$
$P^{(t)} = \beta^{(t)} \frac{c}{V^{(t)}} - P^{(0)}$
/* Compute face normals */
$\mathcal{N}_{\mathbf{f}} = \frac{A_{\mathbf{f}} B_{\mathbf{f}]} \times A_{\mathbf{f}} C_{\mathbf{f}}}{\left\| A_{\mathbf{f}} B_{\mathbf{f}]} \times A_{\mathbf{f}} C_{\mathbf{f}} \right\|}, \quad \forall \mathbf{f}(A, B, C) \in \mathcal{F}$
/* Step 1: Compute pressure forces at each vertex */
$\mathbf{F} = \mathbf{0}$
for $\mathbf{f} \in \mathcal{F}$ **do**
$\mathbf{C} = \texttt{count\_vertices}(\mathbf{f})$
$A = \texttt{area}(\mathbf{f})$
$p = P^{(t)} A \frac{1}{\mathbf{C}}$
$\mathbf{F}_j = \mathbf{F}_j - \mathcal{N}_{\mathbf{f}} p, \quad \forall j \in \mathbf{f}$
end
return F

end

For the force computations of Algorithm 2 we need the set of faces that make up the surface of the air-chamber. These faces can be found using multiple different approaches, but we recommend using the signed distance map representation of the air-chamber, as the faces to include will then be those at the boundary of the distance map. Our model allows for the addition of multiple air-chambers, as each chamber will simply have its own set of faces, initial volume, and even initial pressure.

Implementing the Pressure Model in NVIDIA Flex

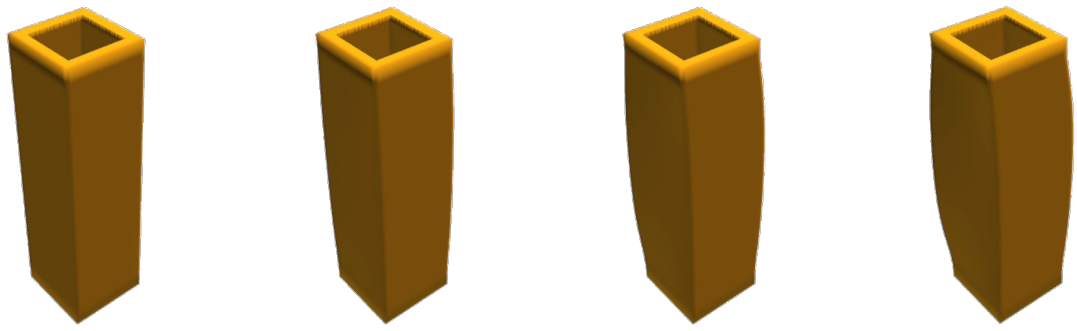
As we will be implementing our pressure force model in Flex, we again need to deal with the fact that we can only interact directly with the particles placed on the surface mesh. We would have liked to implement the pressure forces as an implicit part of the Flex solver, but we have had to settle for an explicit scheme as in the cable force implementation, due to limited exposure of the solver. However, as our air-chambers are specified as a subset of the surface mesh, Flex enforces no limitations on air-chamber design, contrary to how the cable embedding had to be constrained. Observe in Figure 5.3 an example of a simple robot with a single air-chamber. Here, the surface of the air-chamber is visualised in blue.



(a) A simple robot. (b) The wireframe of the robot. (c) The surface of the air-chamber visualised in blue.

Figure 5.3: An example of a simple robot with a single air-chamber and the triangulation of its surface. The air-chamber of the robot is visualised in blue.

The robot is fastened at the top and bottom, which means that inflating the robot using our pressure force model should cause the rectangular robot to approximate a cylindrical shape. The result of employing the Flex implementation of our pressure force model to create an inflation can be seen in Figure 5.4.



(a) . (b) . (c) . (d) .

Figure 5.4: A simple soft robot being inflated over time. The air-chamber of the robot can be seen in Figure 5.3 above, and the top and bottom of the robot are immovable.

In our simulations, the pressure inside a robot should not be able to rise or fall instantaneously,

as this is not how we would be able to control a pneumatic actuator in a physical experiment. However, as we currently do not have any sophisticated pressure regulators, we have chosen to let pressure change by a factor of 0.01 in each simulation step. This is a rather fast rate of change, but it seems acceptable in terms of the simulations it produces.

A more critical part of our simulations is that, we need to compute the volume of each air-chamber at the beginning of every simulation step. In [MO03] volumes were computed using bounding objects, which yields a very rough estimate of the current volume. As our robots are on a millimeter scale, computing correct volumes are crucial if we are to observe behaviour that resembles the real world. Instead of using bounding boxes, we have chosen to approximate the volume of our chambers, as described in [ZC01], by accumulating the signed volume of each elementary tetrahedron of the inner surface - that is, for each triangle on the air-chamber surface we construct a tetrahedron by connecting each of its vertices to the centre of the air-chamber. Observe in Figure 5.5 a sketch of this approximation scheme in 2D.

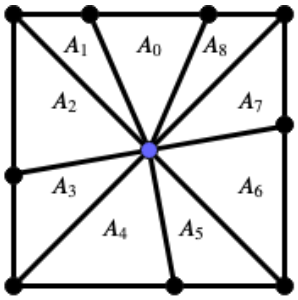


Figure 5.5: The 2D equivalent of our volume approximation scheme. To compute the area of the entire object we would sum the area of each elementary triangle element, providing a good estimate of the total area of the chamber.

However, since our chambers will not be closed objects, we will often get a less accurate approximation such as in the example sketched in Figure 5.6.

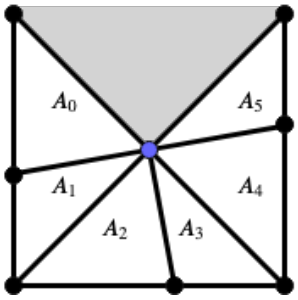


Figure 5.6: The 2D equivalent of our volume approximation scheme for an non-closed mesh. Due to the mesh being non-closed, the approximated area is quite different from the total area we would want to have measured. The part of the chamber we did not measure is marked in grey.

Thus, this method does not accurately compute the volume of our air-chambers, but due to our air-chambers usually being symmetrical, the ratio given by $\frac{V^{(0)}}{V^{(t)}}$ should reflect the real behaviour of our pressure-driven robots fairly well.

5.3 Experiments

5.3.1 Inflation of a Hollow Cuboid

Introduction

We want to examine if our pressure force model produces correct forces during the inflation of a deformable body. We will validate our results based on the volume and internal pressure of the object at the end of the simulation. To verify that our pressure model is not behaving completely unrealistic, we will also perform a visual inspection of the robot during actuation

Procedure

For this experiment we will be inflating the simple pressure driven robot from Figure 5.4. The tetrahedral mesh of the robot can be found in Figure 5.7.

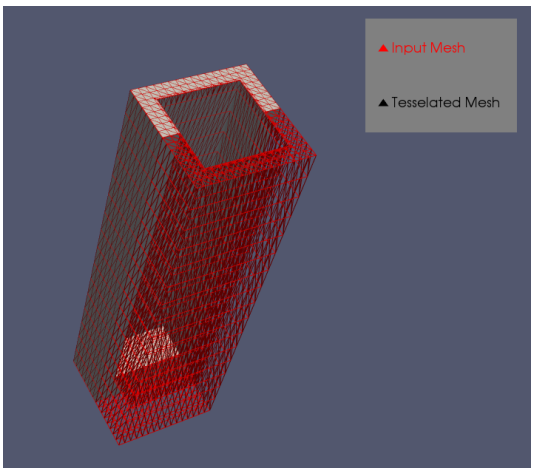


Figure 5.7: The tetrahedral mesh of the robot from Figure 5.4.

To emulate our material of choice for soft robot fabrication - Ecoflex 00-50 - we will use the material properties from Table 5.1, which are based on the work of [Ati12; Smo; Gas+19].

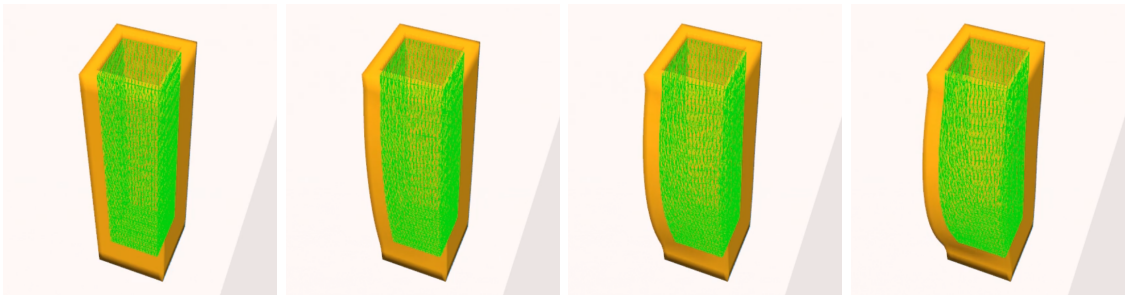
Young's modulus (E)	0.2642 MPa
Poisson's ratio (ν)	0.4999
Density (ρ)	1.07 gram / cc

Table 5.1: Values used to approximate the behaviour of Ecoflex 00-50 in the simulations.

During our simulation, the top and bottom of the robot will be locked in place, such that only the parts of the robot surrounding the air-chamber can expand. The pressure inside the robot will be increased by a factor of 1.10, which should lead to an increase in volume of 10%. The initial volume of the air-chamber within the cuboid was measured at $2.36531 \cdot 10^{-5} m^3$, which means that at the end of the simulation the volume should have increased to $2.60184 \cdot 10^{-5} m^3$ and the simulation should be at an equilibrium - i.e. the internal pressure should now be exactly equal to the initial pressure. For these simulations, gravity was not applied to the robot, as we only wish to examine the isolated behaviour of the pressure forces, and the initial pressure was 1 atmospheric pressure, given by $101 \cdot 10^3$ Pa.

Results

Observe in Figure 5.8 the hollow cuboid robot during actuation.



(a) (b) (c) (d)

Figure 5.8: The hollow cuboid being inflated. Frame (a) is at the beginning of the simulation, and Frame (d) is at the end.

From Figure 5.8 we notice that the robot does expand, but also that it does not expand uniformly. Instead, the robot seem to expand primarily in one direction, while the remaining three sides of the cuboid are more or less untouched by the pressure forces. In Figure 5.9 the change in volume and pressure can be found as functions of time.

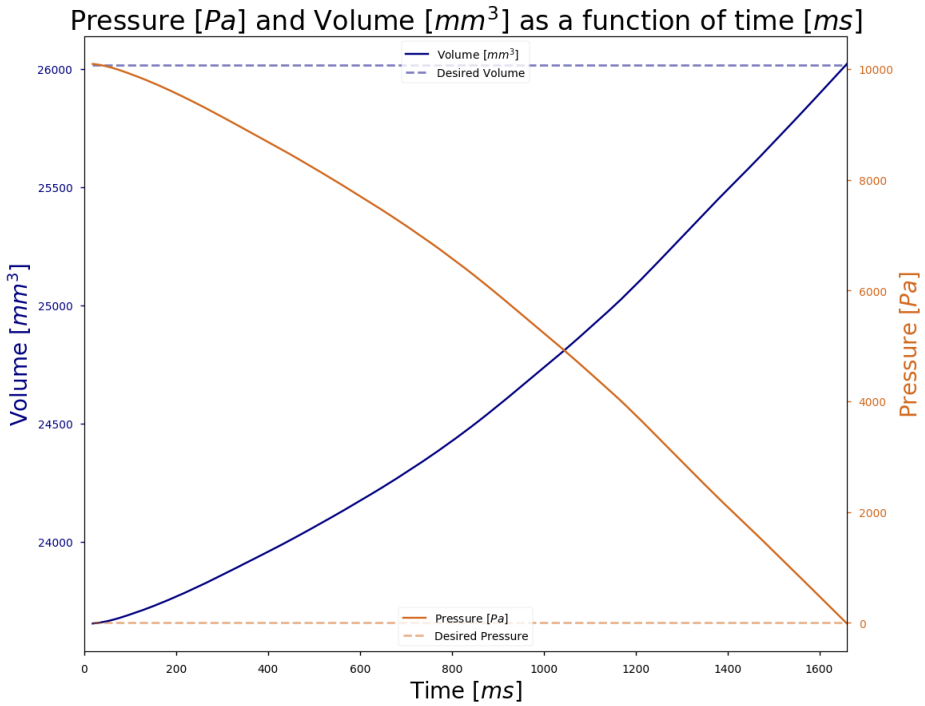


Figure 5.9: The evolution of volume and pressure during the simulation. The dotted lines indicate the desired values in terms of volume and pressure at the end of the simulation. In this plot, a pressure of 0 Pa corresponds to the initial pressure of $101 \cdot 10^3$ Pa.

From these results we see that the volume and pressure act as anticipated, even though we did not observe the expected deformations in Figure 5.8. This indicates that the pressure forces are computed correctly, but also that the forces are distributed improperly across the surface of the air-chamber. Examining the air-chamber mesh of the hollow cuboid, we notice that the

triangle elements of our mesh appear to be elongated, and therefore, contain sharp angles. Upon expansion, the normal of each element are used to compute the positional update we apply to the particles on the mesh. However, this means that errors will accumulate quickly due to the normals of the triangles reacting abruptly to particles moving throughout the simulation, and we believe this might be the reason we observe unexpected behaviour during the simulation. In later experiments we will investigate the relation between mesh quality and pressure force simulations more thoroughly to either support or reject this hypothesis.

5.3.2 Deflation of a Hollow Cuboid

Introduction

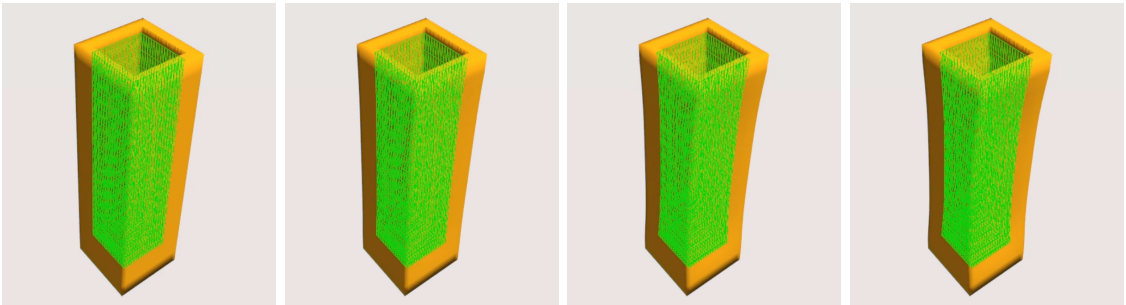
Our pressure force model should not only be able to create inflations in our soft robots, but also deflations. Therefore, we wish to investigate if our pressure force model produces correct forces during the deflation of the same deformable body that we inflated in Experiment 5.3.1. We will validate our results based on the volume of the hollow cuboid at the end of the simulation, as well as the pressure inside the robot at that time. To verify that our pressure model is not behaving completely unrealistic, we will also do a visual inspection of the robot during actuation

Procedure

For this experiment we will be using the same robot and the same material properties as in Experiment 5.3.1. However, instead of increasing the pressure inside of the robot by a factor of 1.10, we will decrease the pressure, setting $\beta^{(t)} = 0.975$ for the entirety of the simulation. The reason that we will not be lowering the pressure any further, is to reduce the risk of the inner walls of the robot collapsing. At the end of our simulations, we should end up having an internal volume of $2.30618 \cdot 10^{-5} m^3$ - i.e. a reduction of the initial volume of $2.36531 \cdot 10^{-5} m^3$ by 2.5% - and as this volume will result in an equilibrium in the pressure forces, we should have an internal pressure exactly equal to the original pressure. For this experiment the initial pressure was 1 atmospheric pressure, or $101 \cdot 10^3$ Pa.

Results

Observe in Figure 5.10 the hollow cuboid during actuation.



(a) (b) (c) (d)

Figure 5.10: The hollow cuboid being deflated. Frame (a) is at the beginning of the simulation, and Frame (d) is at the end.

From Figure 5.10 we observe that the robot behaves as expected, as all walls of the air-chamber are being drawn towards the centre of the cuboid. In Figure 5.11 the change in volume and pressure can be found as functions of time.

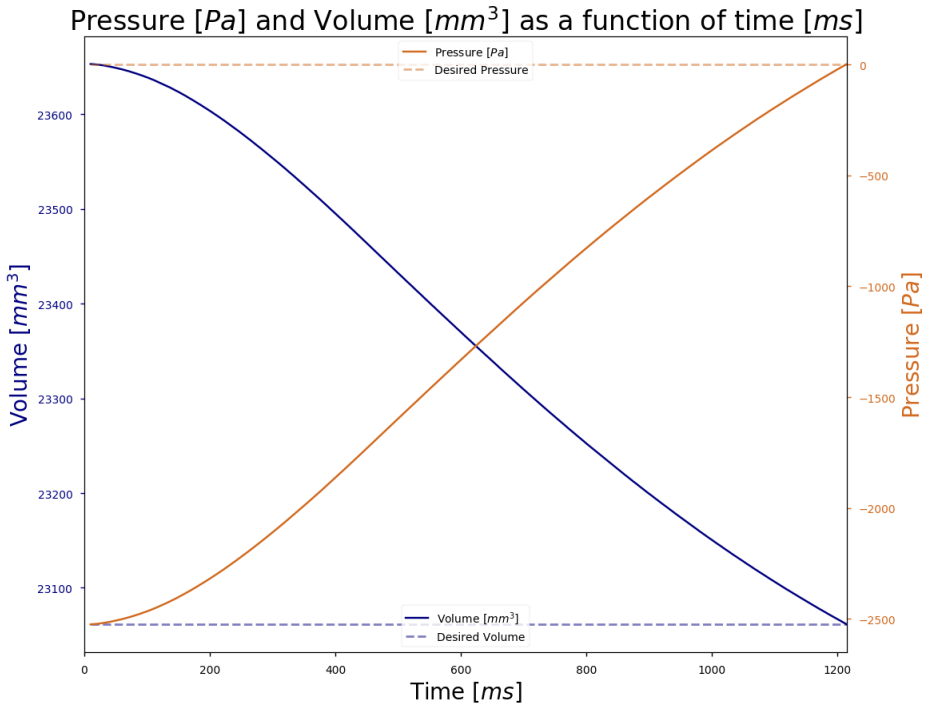


Figure 5.11: The evolution of volume and pressure during the simulation. The dotted lines indicate the desired values in terms of volume and pressure at the end of the simulation. In this plot, a pressure of 0 Pa corresponds to the initial pressure of $101 \cdot 10^3$ Pa.

The measurements were as expected and the robot exhibited no unforeseen behaviour during actuation. This could indicate that during inflation, the pressure force computations are more vulnerable to errors accumulating due to bad meshing, than they are during a deflation. However, as the volume and pressure behave as expected for both of the pressure experiments, we will assume that the pressure model works as intended for the purpose of actuating our soft robots.

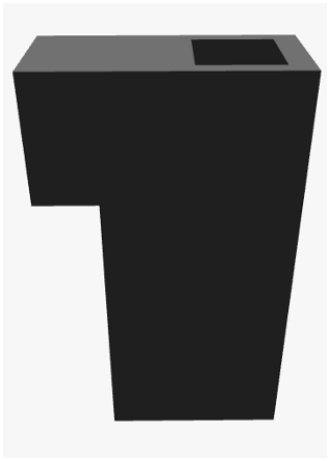
Chapter 6

Examining the Gap between Simulations and Reality

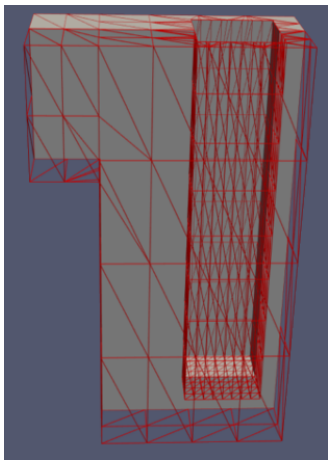
Now that we are able to simulate cable and pressure forces, we need to investigate whether or not the simulations match the real world. We will examine how we can explore the workspace of robot prototypes, and compare the simulated robot motion to physical robots responding to the same control signals. In recent work on the topic of robot control, it is common to compare kinematics [Hao+17], macroscopic errors - such as how much a robot bends [RCD17] - or how well a control policy transfers from simulations to the real world [Sin+19]. We will be collecting quasi-static motion data using the Learning Cube framework, and will compare the motion on a macroscopic scale by overlaying the simulations on real world footage. However, before we can collect and compare this data we need to be able to instantiate robots in Flex and manufacture physical copies of the digital models.

6.1 Exploring Robot Motion in Flex

The first step towards instantiating a robot in Flex is to design a prototype and construct a surface mesh approximating its geometry. Using the design model from the pre-project, we can design and create surface meshes using CSG operations, and verify that the robustness conditions described in Section 2.1 are satisfied, which gives us a strong guarantee that the robot can be fabricated. We then need to create a volumetric mesh by tetrahedralising the surface of our robots, for which we have chosen to use the TetGen [Si] software package, due to its availability and ease of use. Observe in Figure 6.1 an example of how a robot surface mesh is turned into a volumetric mesh. Notice that the robot contains an area without any mass, which we will use as an air-chamber later on.



(a) A robot surface representation.

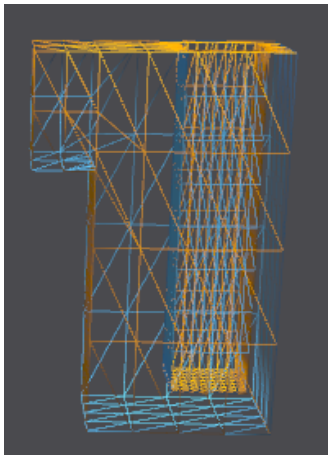


(b) The tetrahedralised robot.

Figure 6.1: An example of a robot surface mesh and its volumetric representation. Here, only half of the tetrahedra are visualised, such that the granularity of the mesh is visible.

Once we have the two representations of our mesh we can instantiate the robot in the Flex simulator, giving it material properties - i.e. Poisson's ratio, Young's modulus, and density - to match the material we want to use later on in the manufacturing process. In Figure 6.2 the robot from Figure 6.1 can be seen, next to the triangulation of its surface mesh.

(a) The robot inside Flex.



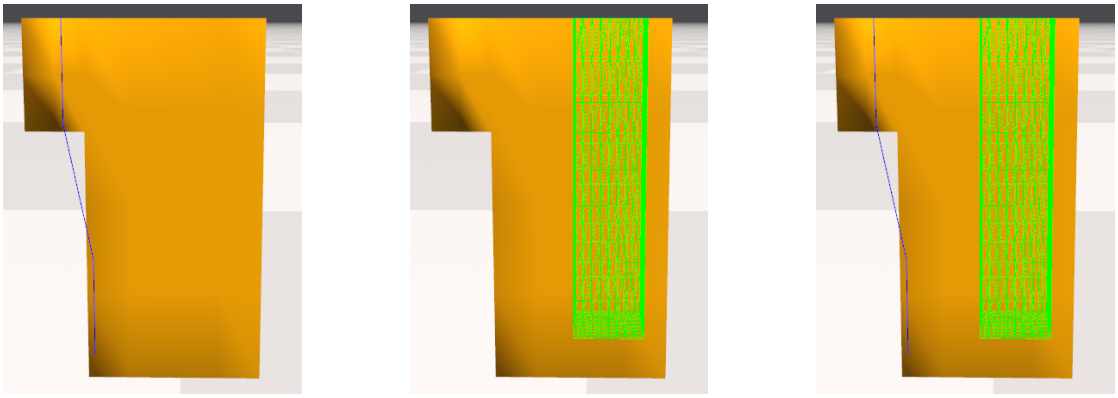
(b) The surface mesh of the robot inside Flex.

Figure 6.2: The robot from Figure 6.1 instantiated in the Flex simulator.

Our robot can contain multiple actuation devices, and our force models place no restrictions on the amount, size or properties of each actuator. However, it should be noted that for each actuation device we add to the robot, the manufacturing process increases significantly in difficulty. To add a cable to a robot we need to specify the coordinates of the attachment and via points of the cable. Currently, each points must lie within the plane of a triangle on the surface mesh, but in future iterations of the cable model, the cable points should be projected onto the mesh if they lie inside or outside of the surface, as this would allow for more leniency in the specification of the cable. Finally, because cables are modelled as piecewise connected springs, we need to specify

their stiffness as detailed in Chapter 4.

Defining an air-chamber is a bit more tricky - depending on the geometry of the chamber - and we recommend using the signed distance map representation of the chamber to find the triangle elements of the chamber surface. The elements will lie on the boundary of the signed distance map, which means that it provides a convenient way to acquire the elements of the surface. Observe in Figure 6.3 examples of how cables and air-chambers can be added to a robot. Here, the cable is visualised as blue connected linear segments, while the triangle elements of the air-chamber are highlighted in green.



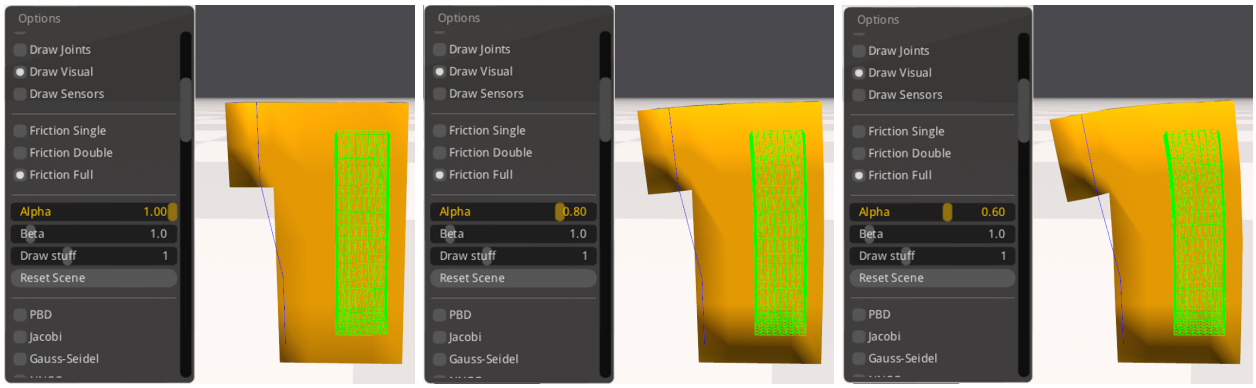
(a) The robot inside Flex with a cable embedded.

(b) The robot inside Flex with a single air-chamber

(c) The robot inside Flex containing both an air-chamber and a cable.

Figure 6.3: The robot from Figure 6.1 instantiated in the Flex simulator with various actuation devices added.

To examine the workspace of a robot we need to be able to control the actuation devices inside Flex. For cable-driven robots we want to control the rest length of each cable, and for pressure-driven robots we want to be able to regulate the pressure inside each air-chamber. Given a robot, such as the one from Figure 6.3, the control parameters, would be controllable from within the Flex simulator, as shown in Figure 6.4.



(a) (b) (c)

Figure 6.4: An example of how the control parameters of a robot can be changed to explore the robots configuration space in real-time. In this example, the control parameter of the blue cable, α, is changed from 1.0 to 0.8 and then from 0.8 to 0.6.

6.1.1 Importance of Meshing

We are implementing our actuation models in a simulator using finite elements, which means the quality of those elements - i.e. tetrahedra and triangles - influence the fidelity of the simulations. In general for finite element methods, we want equilateral triangles of roughly the same size, and want to avoid unbalanced elements, such as very skewed triangles [She02]. Examining the robot mesh from Figure 6.2b, we observe a clear violation of these principles, since the triangle elements of the air-chamber are much smaller than the elements of the rest of the robot body. In [Mül+07], the position-based dynamics scheme - which XPBD is based on - is presented, and it is highlighted that to ensure stable self-collision detection and response, the triangle elements of a mesh must be of approximately the same size. Thus, before even applying our actuation models we reduce the accuracy of the simulations, by using a low quality mesh. In our experience coarse meshes seem to be favourable for cable force simulations, in terms of both efficiency and visuals, contrary to how we would normally expect our simulations to behave. In [Ber+19] coarse meshes were used to simulate cable-driven locomotion of a soft 'foam' robot. The authors stated that they experienced the phenomenon *numerical stiffening* - as described in [Che+17] - which means that the coarse mesh acts more stiffly than its more fine-grained counterparts, even though the elements were subject to the same material properties. In both the model from [Ber+19] and in our own model, cables are modelled as unilateral springs, which means that the force computations are incredibly stiff. It therefore seems possible that the coarse elements of a mesh - which are made stiffer by the numerical stiffness factor - are better suited for integration with the stiff cable forces. However, extensive experiments are needed to verify this, which we leave for future work. Besides from making the robot behave more stiffly than it is supposed to, coarse mesh elements reduce the fidelity of simulating bending in our robots. In Experiment 6.3.1 we will examine the combination of both small and large elements as a solution to the problem of both ensuring realistic bending and visually satisfying cable forces, when we ignore issues such as self-collisions.

In our pressure force model, we compute the forces affecting the inside of an air-chamber, which means that we want the mesh to be as fine as possible. This is due to the fact that if the mesh is too coarse, we have a bad approximation of the surface of the air-chamber, which means that our simulations will not be accurate. While multiple different actuators can be placed inside a single robot, we recommend not doing so with our current force models, as the quality of each simulation relies on different properties of the elements of the mesh.

6.2 Fabricating Physical Copies

We can now simulate our soft robots before manufacturing physical models and explore how they behave during actuation, but so far we have no guarantees that the simulated motion resembles the real world behaviour of robots. However, before we can investigate the gap between simulations and reality, we need to fabricate physical copies of the digital robots.

The robots we will be fabricating are dubbed 'Hammerbot', 'Spinebot' and 'Bubblebot'. The 'Hammerbot' and 'Spinebot' are cable-driven, while the 'Bubblebot' robot is actuated using pneumatics. Cable shelves were added to the cable-driven robots - as described in Chapter 2 - such that cable sheathing can be safely placed without compromising the removal process.

For all of our robots, the digital models were created such that they satisfy the robustness conditions of Equation (2.8). We chose a required minimum thickness of $\delta = 2$ mm, a minimally

allowed angle of $\theta = 15°$, and for all robots we coose the direction $\mathbf{d} = (0, 1, 0)^T$ as the extraction direction. We selected these values based on our own observations and experience, as well as the results presented in Chapter 2, and the final models that passed the robustness tests can be found in Figure 6.5.

(a) The 'Hammerbot' model. (b) The 'Spinebot' model. (c) The 'Bubblebot' model.

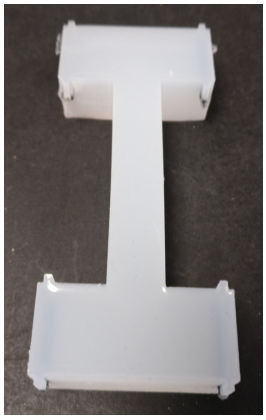
Figure 6.5: The models of the robots that passed the robustness tests.

From these models we created 3D-printable molds containing a single cavity equal to the models from Figure 6.5, using the default printing options of the Ultimaker 2+. Observe in Figure 6.6 the three molds of the 'Hammerbot', 'Spinebot', and 'Bubblebot' robot, respectively.

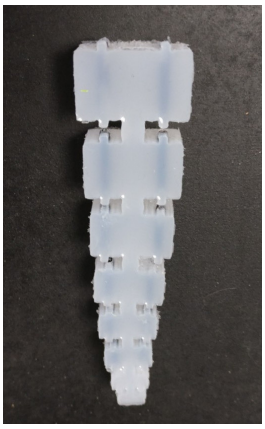
(a) The 'Hammerbot' mold. (b) The 'Spinebot' mold. (c) The 'Bubblebot' mold.

Figure 6.6: The molds corresponding to the robot models from Figure 6.5.

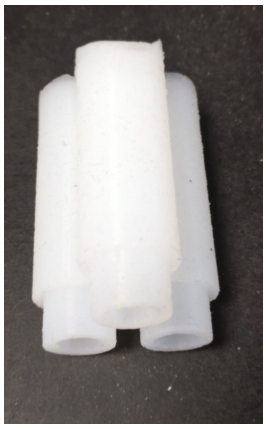
To create the soft bodies of the physical robots we used Ecoflex 00–50 [Smo], and black plastic straws, with a 2 mm diameter, as cable sheaths. The results of casting the robots and removing them from their molds can be found in Figure 6.7.



(a) The 'Hammerbot' robot.



(b) The 'Spinebot' robot.

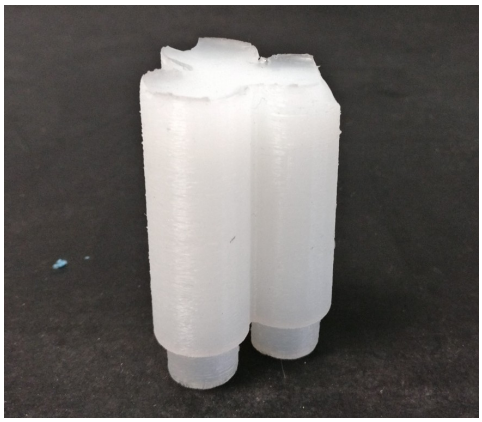


(c) The 'Bubblebot' robot.

Figure 6.7: The physical copies of the robots. Notice the black cable sheaths inside the 'Hammerbot' and 'Spinebot'.

As is evident from Figure 6.7 all robots were removable from their molds, but removing the 'Bubblebot' robot required significantly more force than the 'Hammerbot' and 'Spinebot'. This resulted in both the robot and its mold breaking. The broken mold and a close-up of the damaged robot can be seen in Figure 6.8.

(a) The broken mold of the 'Bubblebot' robot.



(b) Ruptures in the soft body of the robot can be seen near the top.

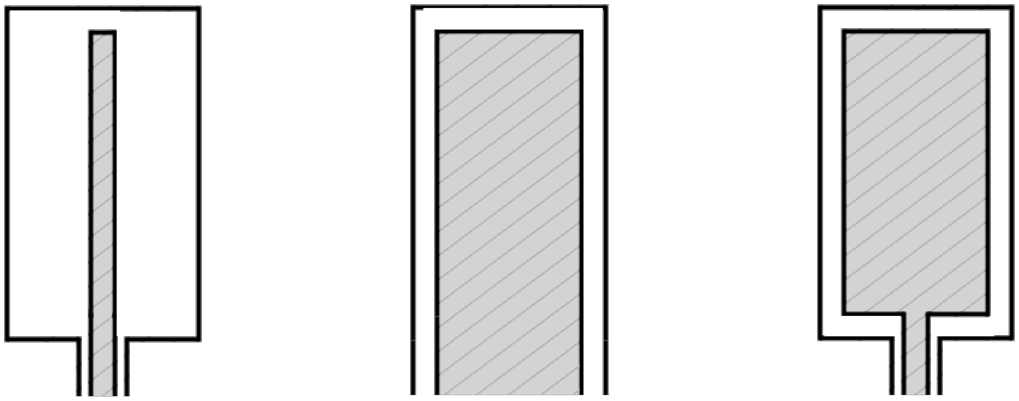
Figure 6.8: The broken 'Bubblebot' mold and robot. The model of the robot satisfied the robustness conditions of Equation (2.8), but during the extraction process the soft body was 'chipped' at several points around the top facet. The mold had the tubes within it - corresponding to the air-chambers of the robot - torn free from the base during removal.

Now, the 'Bubblebot' robot passed the robustness test, but during extraction all three air-chamber cylinders ended up breaking and the robot ruptured in multiple locations. The risk of the mold fracturing can be reduced by increasing its density, but this should have no effect on the damage done to the soft body during removal. The parameters we chose for the robustness test were more relaxed compared to the settings recommended in the pre-project. However, examining the ruptures in Figure 6.8a reveals that the weak points of the soft body were located at the densest part of the robot. Thus, more restrictive settings would not have made the extraction process more

robust, and instead indicates that the guarantee provided by the robustness conditions was not as strong as we perceived it to be.

In the case of the 'Bubblebot' robot, the vacuum created in the tall mold made the extraction process extremely difficult, which highlights a weakness in the robustness conditions: the relationship between width, height and length of a robot is not taken into account. Thus, a design such as the one for the 'Bubblebot' robot could have been invalidated in the design phase, by verifying whether or not the dimensions of the robot conformed to the limitations of the single-mold single-cast manufacturing process. However, in terms of producing working and interesting pressure-driven robots, limiting the dimensions of the robot would make the design phase significantly more difficult. Recent trends in pressure-driven robot designs indicate that air-chamber geometry is crucial in terms of controlling soft robots using pneumatics [Hao+17; Gal+16; Gli+18; She+11], and by placing harsh limitations on the size of the soft robots, it becomes more demanding of the designer to fashion proper air-chambers.

The most common way to produce pneumatically actuated robots seem to be using techniques such as soft lithography, multi-part molds, and multi-step casting [Hao+17; Gal+16; Gli+18; She+11; Ili+11], as these methods impose very few limitations on the design of air-chamber geometry. Our model lacks the ability to create intricate air-chambers, as only very basic designs satisfy the first robustness condition. In fact, our model not only limits the action space of our soft robots, but also the fidelity of the manufactured robots compared to our simulations. Observe in Figure 6.9 how the first robustness condition limits our options in terms of air-chamber design.



(a) Making the air-chamber smaller to allow for a smaller intake.

(b) Making the intake larger to allow for a larger air-chamber.

(c) An ideal air-intake and air-chamber combination. **Not** possible with our robustness model.

Figure 6.9: Example air-chambers (shaded) with different air-intakes. Figures 6.9a and 6.9b showcases the two options available when the air-chamber needs to satisfy the robustness conditions, and Figure 6.9c shows how we would ideally want our air-intake and air-chamber to be.

The first robustness conditions states that there must exist a direction that solves the mold extraction problem, which means that we cannot have any overhang in our molds. However, this means that the air-intake - i.e. where we will connect our robot to an actuation device - cannot be smaller than the air-chamber itself. For small air-chambers this is not a real issue, as it is fairly easy to connect the robot to an air-supply while creating a tight fit. However, the actuation devices available to us connect to the air-chambers of a robot using silicone tubing with a 5 mm

diameter. Unfortunately, we need our air-intakes to be smaller than the tubes to create an air-tight fit, and even then, more steps need to be taken to ensure that the tubes are not repelled from the robot during actuation. We therefore need very small air-intakes, but as this also restricts our air-chamber size, it becomes difficult to design a functional robot, for which we can attach our actuation device without causing unwanted deformations in the soft body. Observe in Figure 6.10 how our current way of attaching our pneumatic actuator twists the body of one of our robot near the air inlets.

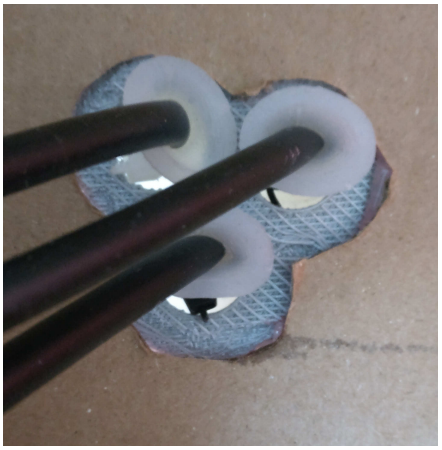


Figure 6.10: The tubing of our pneumatic actuators fixed to a soft robot using plastic strips. Notice how the robot is 'squished' because of the strips.

Similarly to how the air-chambers of our pressure-driven robots are limited by the robustness conditions, so is the placement of cables, as we cannot place cables directly above each other without violating the robustness conditions. Observe in Figure 6.11 how we would be able to place cable sheaths - given that the sheaths are somewhat flexible - if we were to disregard the robustness conditions.

Figure 6.11: An example of a section of a mold (dark cyan) with small 'nooks' where cable sheathing (light grey) can be placed. This allows for cables to be placed directly above each other.

However, for our cable-driven robot designs, we have observed that the single-cast approach has been able to reliably produce robots without damage being done to either robot or mold. This indicates that the robustness conditions are well suited for verifying the feasibility of cable-driven

designs, which might stem from the fact that this type of robot seem to be better suited for single-cast molding than their pressure-driven counter-parts.

To conclude, the observations we have made in this thesis seems to indicate that the feasibility model provides some guarantee on the robustness of a design - especially for cable-driven robots - but currently it is too unreliable to be truly useful. In future work, it might be interesting to verify the robustness of a design by simulating the extraction process. This would allow us to measure the forces we need to apply during extraction, as well as examine the stress that the robot experiences. This might make it possible to provide a better estimation of the feasibility of a robot design, as we would know if the soft body would be able to handle forces needed to complete an extraction.

6.3 Experiments: Exploring the Action Space of Soft Robots

For all of the following experiments, the material properties of the soft bodies can be found in Table 6.1 and are based on the results of [Ati12; Smo; Gas+19].

Young's modulus (E)	0.2642 MPa
Poisson's ratio (ν)	0.499
Density (ρ)	1.07 gram / cc

Table 6.1: Material properties of Ecoflex 00-50 used for all experiment in this chapter.

The material used for the soft body in Flex was given a small factor of 10^{-8} as damping coefficient. This makes the solver able to remove some of the energy from the system, similarly to how it is detailed in [Mül+07].

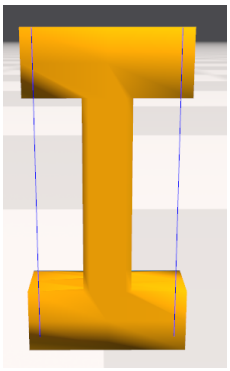
6.3.1 Exploring the Action space of a Simple Cable-driven Soft Robot

Introduction

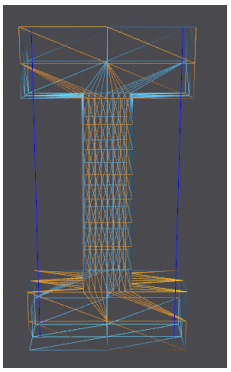
In this experiment we want to examine how we can explore the action space of a simple cable-driven robot instantiated in the Flex simulator, as well as the effect of a combination of coarse and fine mesh elements on the fidelity of the robots behaviour.

Procedure

For this experiment, we will be sampling the configuration space of a simple soft robot, with parts of its soft body having different levels of granularity in their surface triangulation. The robot - the 'Hammerbot' - can be found in Figure 6.12 next to a visualisation of the triangulation of its surface mesh.



(a) The 'Hammerbot' robot we will use for this experiment.



(b) The wire frame - i.e. mesh - of the 'Hammerbot'.

Figure 6.12: The simple robot we will actuate in this experiment. The robot contains two cables, visualised as blue line segments.

The robot is equipped with two cables, which we will actuate both individually and in concert. The cable parameters are given by

$$k = 2000$$
$$b = 0.001$$

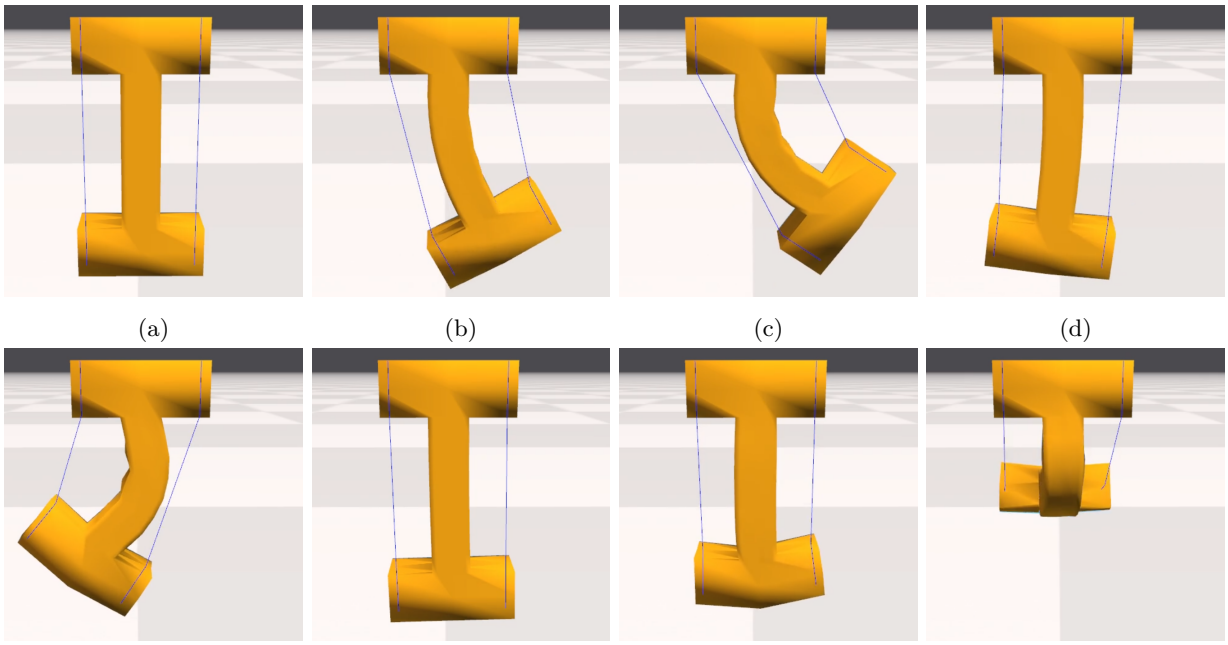
where k is the stiffness coefficient and b is the damping factor. The goal of this experiment is to discover any flaws in the simulation such as self-intersection, inversion of triangles, and visually undesirable deformations. Each of the cables has a rest length of exactly 100 mm, and through the simulation the robot will be subject to a constant gravitational force of -9.8 in the y-direction (downwards)· Finally, the top surface of the robot is locked in place, to let the robot be able to hang in gravity.

Results

In Figure 6.13, still images of the robot during actuation can be found, with α values corresponding to the ones found in Table 6.2.

Frame	α_0 (left cable)	α_1 (right cable)
Frame (a)	1.00	1.00
Frame (b)	1.20	0.82
Frame (c)	1.20	0.70
Frame (d)	0.91	1.01
Frame (e)	0.72	1.01
Frame (f)	1.01	1.01
Frame (g)	0.79	0.81
Frame (h)	0.79	0.81

Table 6.2: Table detailing the α parameter values for each frame in Figure 6.13. Notice that the values are the same for Frame (g) and (h) - however, unlike all the other frames these two show the dynamic development of the cable. All other frames were captured when the cables where at their final desired length.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 6.13: The 'Hammerbot' robot during actuation. See Table 6.2 for the α values of the two cables in each frame.

In this experiment we saw visually pleasing simulations of our robot during actuation, indicating that the simulations can indeed be used to explore the action space of cable-driven robots. However, in Frame (e), we observed some weird behaviour in the beam connecting the two 'hammerheads' which is evidence that we need an even finer mesh to properly simulate smooth bending components. For a video of the 'Hammerbot' being actuated please refer to `mystiking.github.io`.

6.3.2 Exploring the Action space of an Advanced Cable-driven Soft Robot

Introduction

In this experiment we want to explore the configuration space of one of our more advanced soft robot designs. The elements of this robots mesh vary in size, and we want to examine the effects of this on the simulations.

Procedure

For this experiment we will sample the configuration space of a robot consisting of eight connected joints in decreasing sizes. The robot, called the 'Spinebot', can be found in Figure 6.14 and is a smaller version of Robot A.2.

(a) The 'Spinebot' robot we will use for this experiment.

(b) The wire frame - i.e. mesh - of the 'Spinebot'.

Figure 6.14: The robot we will actuate in this experiment. The robot contains two cables, visualised as blue line segments.

The robot contains two cables running from the top of the first joint to the end of the sixth joint, on each side of the robot, measuring 80.5 mm. The cable parameters were given by

$$k = 2000$$
$$b = 0.001$$

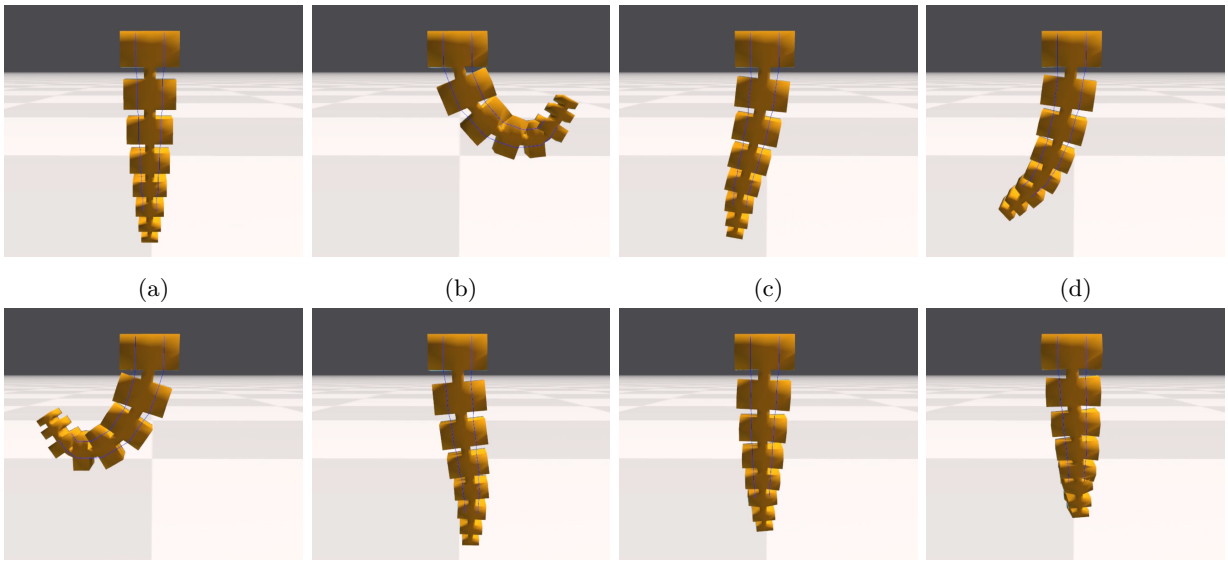
The top surface of the robot is locked in place, meaning that the robot will hang in place for the duration of the simulation. Furthermore, this robot is also subject to gravity as in the previous experiment.

Results

In Figure 6.15 still shots of the robot during actuation can be found, with α values corresponding to the ones found in Table 6.3.

Frame	α_0 (left cable)	α_1 (right cable)
Frame (a)	1.00	1.00
Frame (b)	1.00	0.80
Frame (c)	1.00	1.00
Frame (d)	0.90	1.00
Frame (e)	0.80	1.10
Frame (f)	1.00	1.00
Frame (g)	0.90	0.90
Frame (h)	0.80	0.80

Table 6.3: Table detailing the α parameter values for each frame in Figure 6.15. All frames were captured when the cables where at their final desired length.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 6.15: The 'Spinebot' robot during actuation. See Table 6.3 for the α values of the two cables in each frame.

In this experiment, cable forces again seem to be correct - in terms of direction and magnitude - but we observe self-intersections when the cables are pulled beyond a certain point, as in Frame (b) and (e). For this robot the meshing of the individual joints varied, with small joints consisting of finer elements than the large joints. As we expected, the Flex simulator performed worse in terms of resolving self-intersections, which might be due to the non-uniformity of the triangle elements. For a video of the 'Spinebot' being actuated please refer to `mystiking.github.io`.

6.3.3 Order Dependency of Cable Control

Introduction

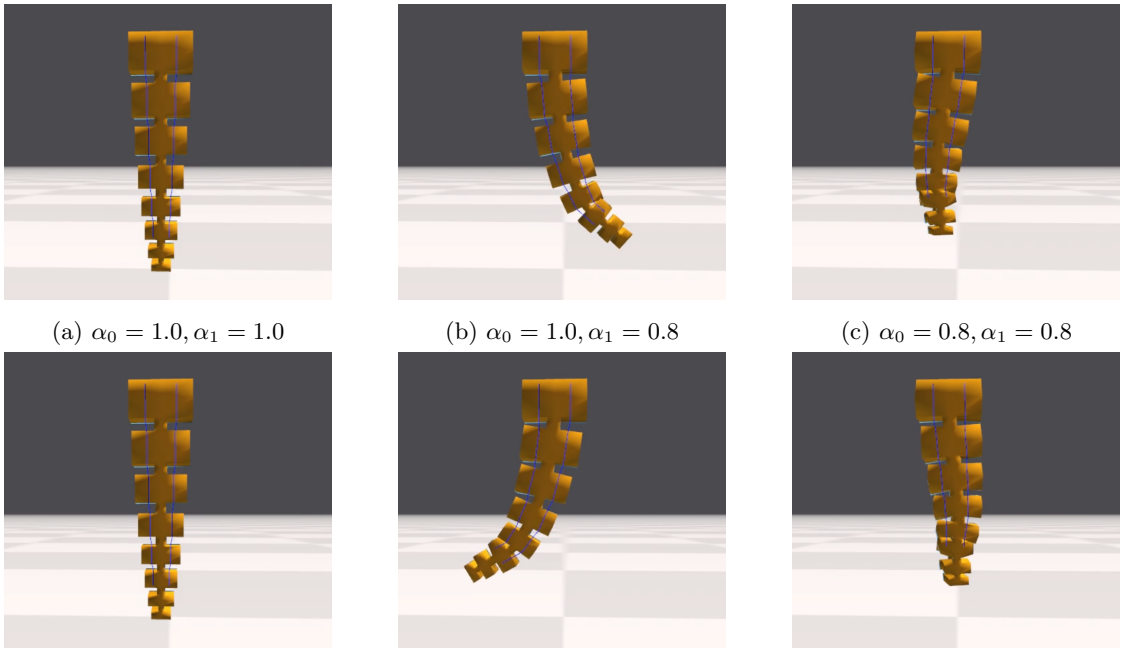
Until now, we have treated our simulations as quasi-static, which entails that for the same set of control parameters, we will always end up in the same configuration. However, as we are using Flex - which is a dynamic simulator - we expect to also observe a dependency between final configuration and the order in which the cables of our robots are pulled.

Procedure

We will examine the behaviour of both the 'Hammerbot' and 'Spinebot', in terms of the configurations resulting from cables being pulled in a specific order. For the 'Spinebot' we will lower the control parameter of one cable to 0.8, letting the simulation arrive at a standstill, and then lower the other parameter to 0.8 as well. We will then repeat the experiment while lowering the parameters in the opposite order. For the 'Hammerbot' we will perform the same experiment, but this time setting the control parameters to 0.75. The robots will both be subject to the same gravitational settings as described in the previous experiments.

Results

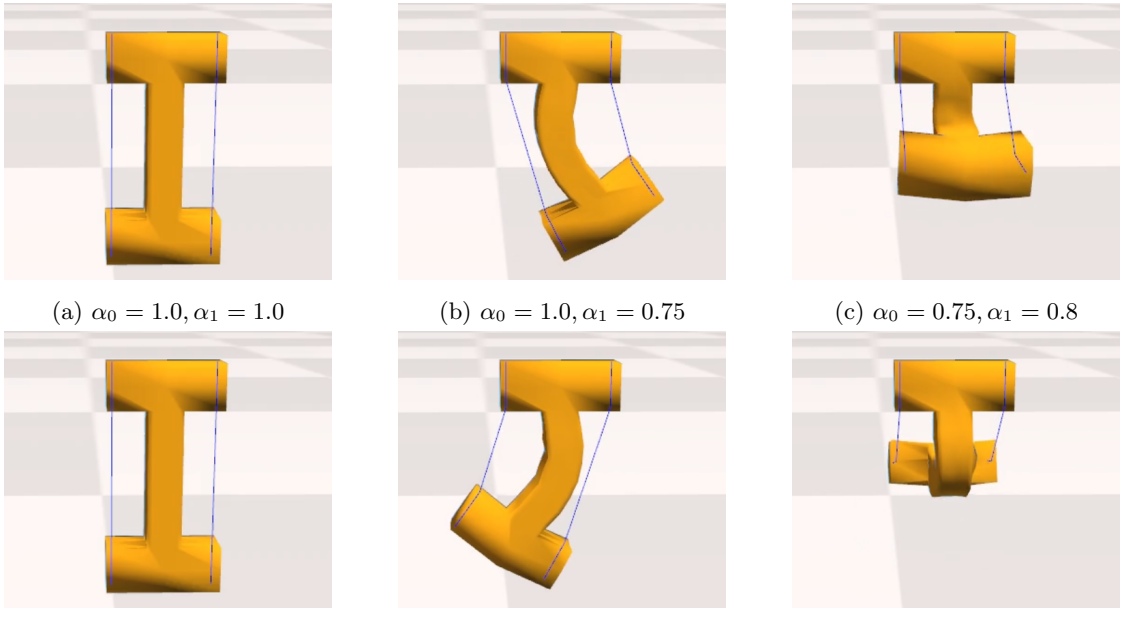
Observe in Figure 6.16 the cables of the 'Spinebot' being pulled in different orders, and in Figure 6.17 the same scenario but with the 'Hammerbot'.



(a) $\alpha_0 = 1.0, \alpha_1 = 1.0$ (b) $\alpha_0 = 1.0, \alpha_1 = 0.8$ (c) $\alpha_0 = 0.8, \alpha_1 = 0.8$

(d) $\alpha_0 = 1.0, \alpha_1 = 1.0$ (e) $\alpha_0 = 0.8, \alpha_1 = 1.0$ (f) $\alpha_0 = 0.8, \alpha_1 = 0.8$

Figure 6.16: The 'Spinebot' robot being actuated. In Frames (a)-(c) we see the right cable being pulled before the left, and in Frames (d)-(f) we see the opposite order. The alpha values of each frame is stated in the sub-caption of the specific frame.



(a) $\alpha_0 = 1.0, \alpha_1 = 1.0$ (b) $\alpha_0 = 1.0, \alpha_1 = 0.75$ (c) $\alpha_0 = 0.75, \alpha_1 = 0.8$

(d) $\alpha_0 = 1.0, \alpha_1 = 1.0$ (e) $\alpha_0 = 0.75, \alpha_1 = 1.0$ (f) $\alpha_0 = 0.75, \alpha_1 = 0.8$

Figure 6.17: The 'Hammerbot' robot being actuated. In Frames (a)-(c) we see the right cable being pulled before the left, and in Frames (d)-(f) we see the opposite order. The alpha values of each frame is stated in the sub-caption of the specific frame.

For both of our robots, we see that the order in which the cables are pulled have a clear effect on their final configuration. In the 'Spinebot' this is characterised as a skewed 'S' shape, with the different orders manifesting as a mirroring of the shape. The 'Hammerbot' displays a much larger difference in final configuration based on the order that the cables are pulled. In Frames (c) and (f) of Figure 6.17 we see that the robot bends into the z-direction - a non-linear motion - with the order of the pulls determining whether it bends forwards or backwards.

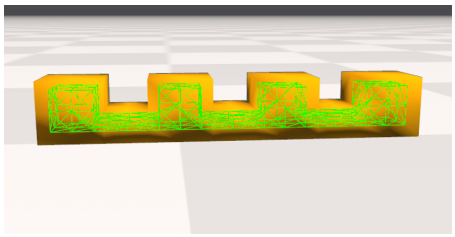
6.3.4 Exploring the Action space of a Simple Pressure-driven Soft Robot

Introduction

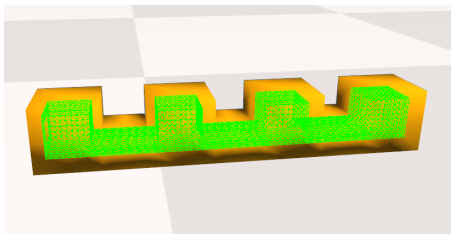
For this experiment we want to sample the configuration space of a pressure-driven robot and examine the effect of a coarse mesh versus a fine mesh for representing the surface of the air-chamber.

Procedure

In this experiment, we will slowly increase the internal pressure of the single air-chamber of the 'Bubblefinger' robot, which can be found in Figure 6.18. We will repeat the same experiment for both of the air-chamber representations and compare the results. Unlike the previous experiments, this design is not compatible with our current fabrication capabilities, and we will therefore not be comparing it to a physical robot.



(a) The 'Bubblefinger' robot with coarse air-chamber surfaces.



(b) The 'Bubblefinger' robot with fine air-chamber surfaces.

Figure 6.18: The robot we will actuate in this experiment. The robot contains a single air-chamber, whose surface is visualised in green. We will be performing our experiment using both of the air-chamber granularities and examine any differences in behaviour.

The initial internal pressure is 10^5 Pa, and during the simulation the pressure will be increased by a factor of roughly 2.5 for both mesh types. So far, our pressure model has appeared to be a lot less stable than our cable force model, and therefore, we will not be applying gravitational forces to this robot. Instead, we would like to isolate the pressure forces such that our results cannot be tainted by external forces, even though this of course means our simulation will not translate to the real world.

Results

In Figure 6.19, the result of actuating the two versions of the robot can be found as still shots from the simulation.

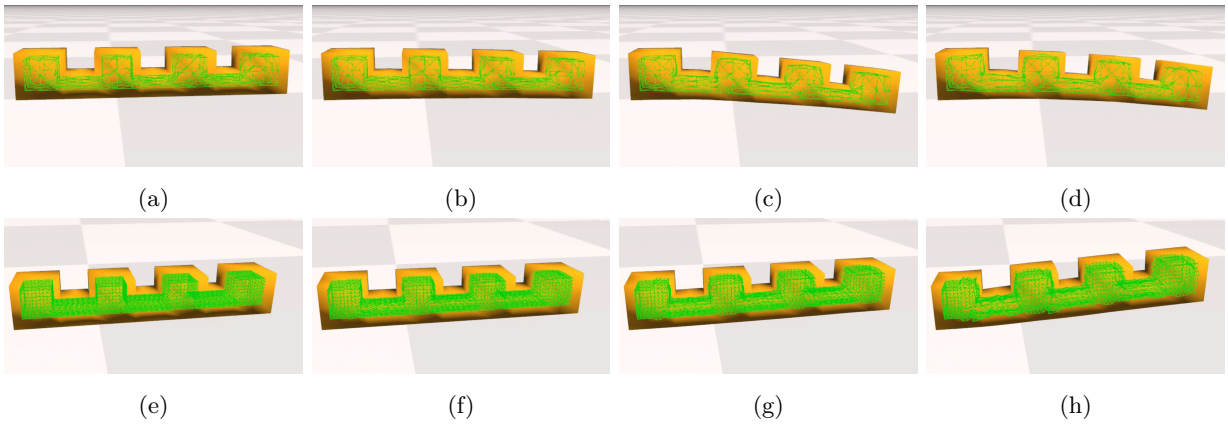
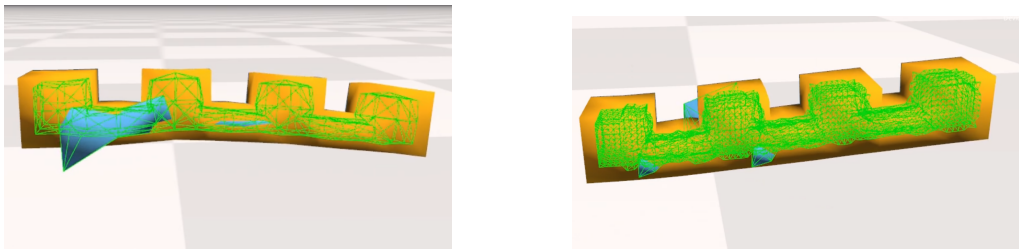


Figure 6.19: The 'Bubblefinger' robots during actuation. Frame (a)-(d) shows the 'Bubblefinger' with the coarse mesh having its internal pressure increased by a factor of 2.57. Frame (e)-(h) shows the 'Bubblefinger' with the fine mesh, using $\beta^{(t)} = 2.45$.

As the wedges of the robot are upwards-facing, we expected the robot to bend downwards as it correctly does in Frame (a)-(d) for the robot with the coarse air-chamber. However, for the robot with the fine air-chamber, we actually observe the opposite of what we expected. Both simulations had to be cut short soon after the final frames, (d) and (h) respectively, as the triangles of both air-chambers began inverting. Observe in Figure 6.20 how the robots 'broke' during the simulations.



(a) The 'Bubblefinger' robot with coarse air-chamber surfaces failing.

(b) The 'Bubblefinger' robot with fine air-chamber surfaces failing.

Figure 6.20: The pressure-driven robot simulations failing. The blue parts of the mesh are the inside of the surface mesh, which means that some of the triangles of the surface have inverted.

These inversions of triangles are typically caused by too large forces acting on the soft bodies of the robots in a too small period of time. This means that in future experiments the time step size of the simulator should be reduced. For the finely meshed robot we also observed unexpected behaviour, and upon closer examination of the air-chamber in Frames (e)-(h), we see that the triangles of the air-chamber are oscillating throughout the simulation. These oscillations seem to indicate that the time discretisation was too coarse given the initial pressure value within the robot. In terms of why only the 'finer' robot behaved unexpectedly, it seems that the increased granularity of the mesh accumulated errors faster than the robot with the coarse mesh. In future experiments, we will therefore need to lower either the time step size of the simulator or the initial pressure within the robot, as well as examine the quality of the mesh in more detail than just the number of tetrahedra elements.

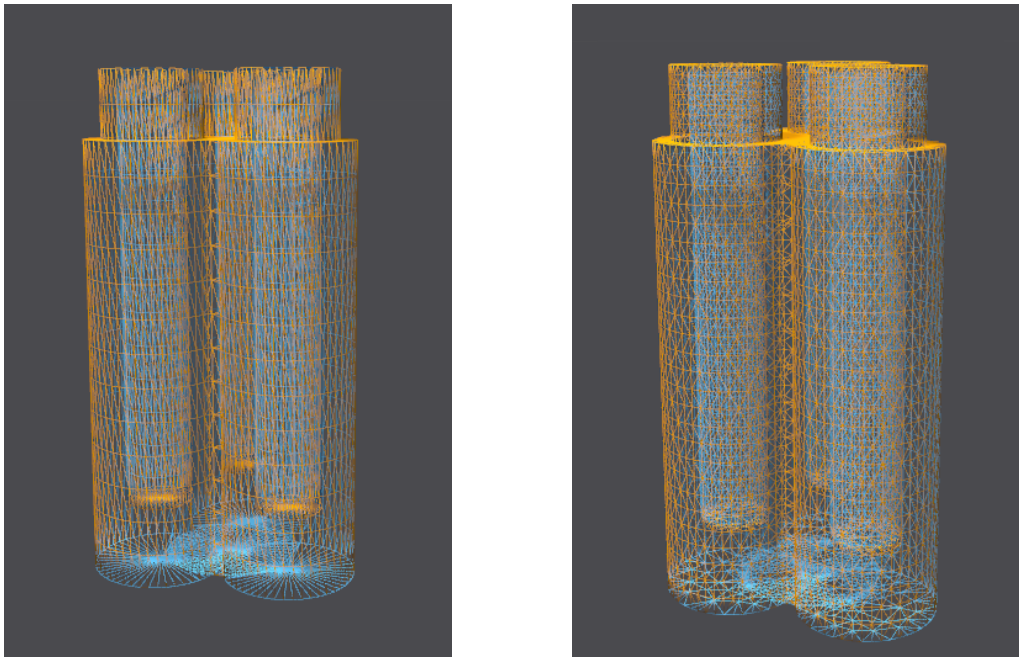
6.3.5 Exploring the Action space of an Advanced Pressure-driven Soft Robot

Introduction

In this experiment we wish to perform a closer inspection of the effect of mesh quality on the fidelity of our pressure force simulations.

Procedure

We will be actuating the 'Bubblebot' robot, using meshes of varying quality. We will inflate one chamber and compare the simulation results, and the mesh quality, between the two models in Figure 6.21. The chamber will have an initial pressure of $101 \cdot 10^3$ Pa - i.e. one atmospheric pressure - and we will inflate the robot such that the internal pressure would be increased by a factor of 5. Furthermore, we reduce the time step size by a factor of 10, which means that we will have $\Delta t = 0.0016$. This should increase the stability of our simulations, and reduce the chance of triangles inverting as we observed in the previous experiment.



(a) A low quality mesh of the 'Bubblebot' robot. (b) A higher quality mesh of the 'Bubblebot' robot.

Figure 6.21: The robot we will actuate in this experiment contains three air-chambers. Notice that the quality of the mesh to the left is lower than the quality of the mesh to the right.

To measure the quality of our meshes we proceed as in [Mis+12] and compare their 'volume-edge ratio', which penalises irregular tetrahedra while favouring regular tetrahedra. The quality measure, Q_{vl}, is defined as,

$$Q_{vl}(t) = \frac{V(t)}{V_{rms}(t)} = 6 \cdot \sqrt{2} \frac{V(t)}{L_{rms}^3}$$

where $V(t)$ is the volume of the t'th tetrahedron, and $V_{rms}(t)$ is the volume of the regular tetrahedra with edge length equal to the root mean square edge length, L_{rms}, of tetrahedron t. We present both the quality measures and the visual simulation result. We aim to support or reject the

hypothesis that a better mesh quality leads to more realistic deformations in the soft body of our robot.

Results

Observe in Figure 6.22 both meshes from Figure 6.21 actuated as described above.

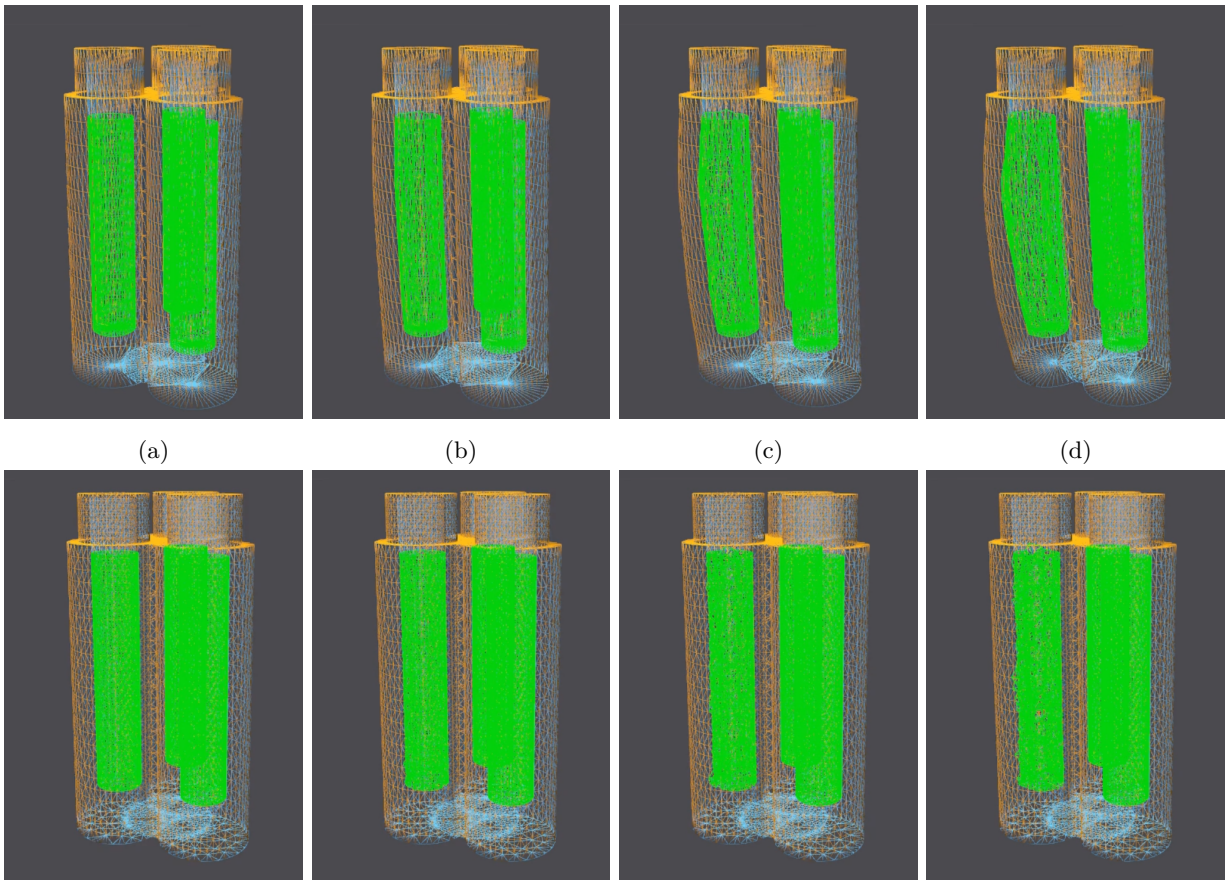
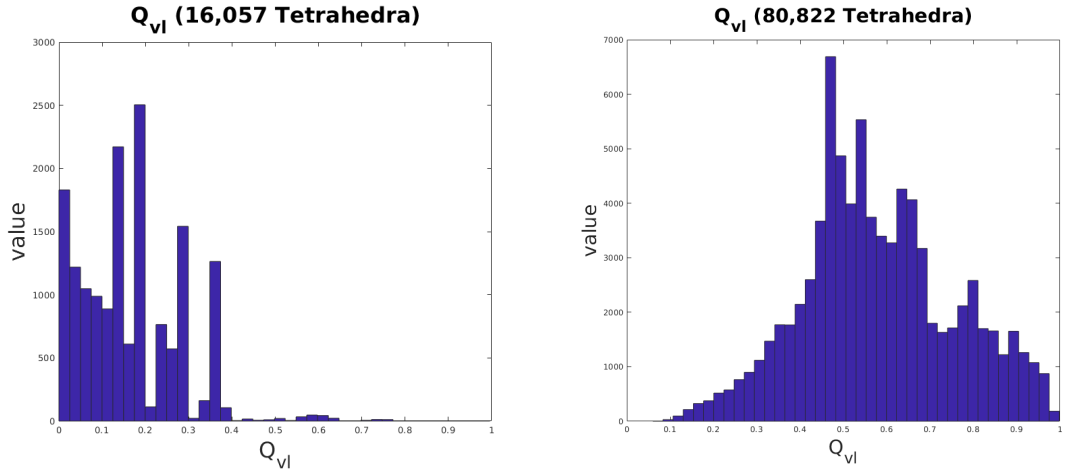


Figure 6.22: The 'Bubblebot' robot during actuation. Frames (a)-(d) shows the wire frame of the robot with the low quality mesh, and Frames (e)-(h) shows the high quality mesh. The air-chambers of the mesh are highlighted in green.

In our low quality simulation we see a small deformation in the direction that we expected, but the deformation is nowhere near our expectations in terms of magnitude as the robot was subject to an inflation that should increase the volume by a factor of 5. However, the high quality simulation performs a lot worse, as we see almost no movement in the robot, and a very jagged deformation near the walls of the air-chamber. Observe in Figure 6.23 histograms of the Q_{vl} measures of the two meshes.



(a) The Q_{vl} measure of the mesh from Figure 6.21a. (b) The Q_{vl} measure of the mesh from Figure 6.21b.

Figure 6.23: The $Q_v l$ measures of the two meshes we simulate. The low quality mesh consisted of 16,057 tetrahedra, while the high quality mesh consisted of 80,822 elements.

From these measures it is clear that the tetrahedra of the high quality mesh are in general more regular than the elements of the low quality mesh. However, as the number of tetrahedra is a lot higher in the high quality mesh, the total number of irregular tetrahedra is also increased. For the low quality mesh 15,843 elements had a Q_{vl} score under 0.5, while the number of elements with a Q_{vl} measure of less than 0.5 in the high quality mesh was 28,865. In other words, we observe almost twice as many irregular tetrahedra.

This seems to indicate that even though the relative quality of a mesh might be higher, we also need to examine the total amount of irregular elements if we want to be able to create realistic pressure force simulations.

6.4 Experiments: Comparison with Physical Robots

We now have some confirmation that our robots behave like we would expect, but to ascertain just how realistic our simulations are, we need to compare them to a 'ground truth' in terms of robot motion. Using the Learning Cube framework, we will capture real robot motion of the robots from Experiments 6.3.1, 6.3.2 and 6.3.5. For our cable-driven robots we will be actuating the cables using the `Mercury` stepper motors described in Chapter 3, which means that we can achieve a high precision replay of the simulated controls. However, for our pneumatically actuated robots we are severely limited as we have no digital controller for the pumps we will be using, which means that we have to resort to using a human operator to 'power' the actuators.

6.4.1 Actuating the Hammerbot

Introduction

In this experiment we actuate a physical copy of the 'Hammerbot'. We aim to investigate any differences between the simulation from Experiment 6.3.1 and how the robot behaves in the real world.

Procedure

The physical robot and the rig we will be using to actuate our robot and capture motion data can be found in Figure 6.24.

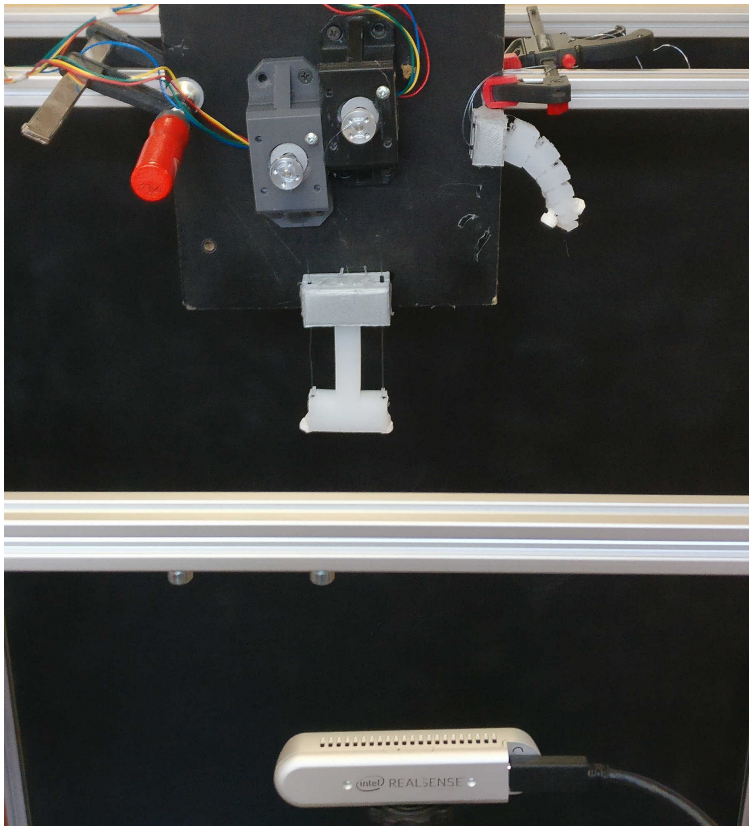


Figure 6.24: The physical set-up of the 'Hammerbot' experiment. Notice that this is a dual rig, which can actuate either the 'Hammerbot' or the 'Spinebot'.

The rest length of each of the cables - the part of the cables inside the robot - measure 100 mm, and in order to distribute forces at the cable end-point we used small white plugs. We actuate the physical robot using the same configurations we applied to the robot in the simulations of Experiment 6.3.1. Observe in Table 6.4 the motor positions we will engage to reach the alpha values from Table 6.2. Both alpha values and motor positions have been added to the table, where each row corresponds to one configuration.

Configuration	α_0 (Left cable)	α_1 (Right cable)	Motor 0	Motor 1
Configuration (a)	1.00	1.00	0	0
Configuration (b)	1.20	0.82	-320	288
Configuration (c)	1.20	0.70	-320	480
Configuration (d)	0.91	1.01	144	-16
Configuration (e)	0.72	1.01	448	-16
Configuration (f)	1.01	1.01	-16	-16
Configuration (g)	0.79	0.81	336	304

Table 6.4: Table detailing the motor settings - in terms of steps - for each configuration we will examine.

Every time the robot has reached a stable point at the end of a configuration we will capture a single RGB image using one of the `Intel Realsense` cameras. To compare our findings to the simulated movement we will present the configurations following from the control parameters from Table 6.4 individually, as well as overlaid on the simulated results from Figure 6.13.

Results

Observe in Figure 6.25 the physical robot configurations corresponding to the control parameters from Table 6.4.

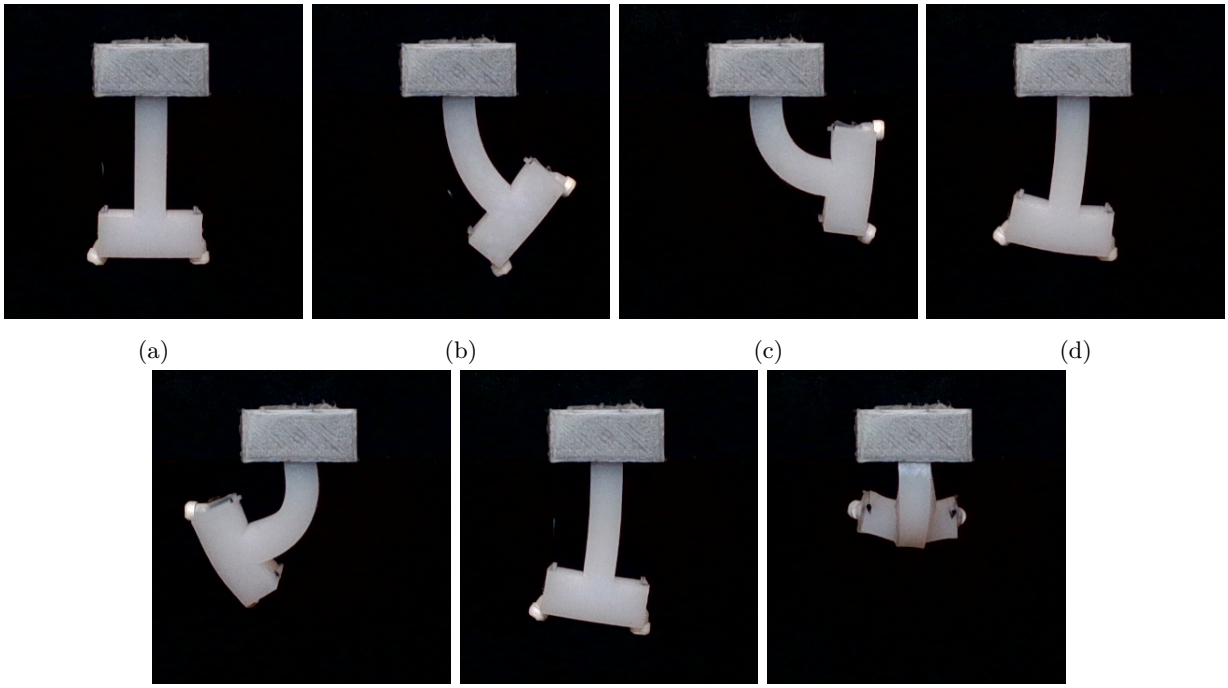


Figure 6.25: The 'Hammerbot' during actuation. The part of the robot encased in gray plastic is being held in place to match the simulated set-up.

From our results in Experiment 6.3.1 we learned that the robot was not able to bend very well due to the coarseness of its mesh, but as we notice in especially Configurations (c) and (e), the robot is able to bend very smoothly in the real world. Observe in Frame (e) that the right cable made contact with the bending part of the robot, causing it to rotate slightly. In Figure 6.26 the physical configurations placed as an overlay on top of the results from our simulations can be found.

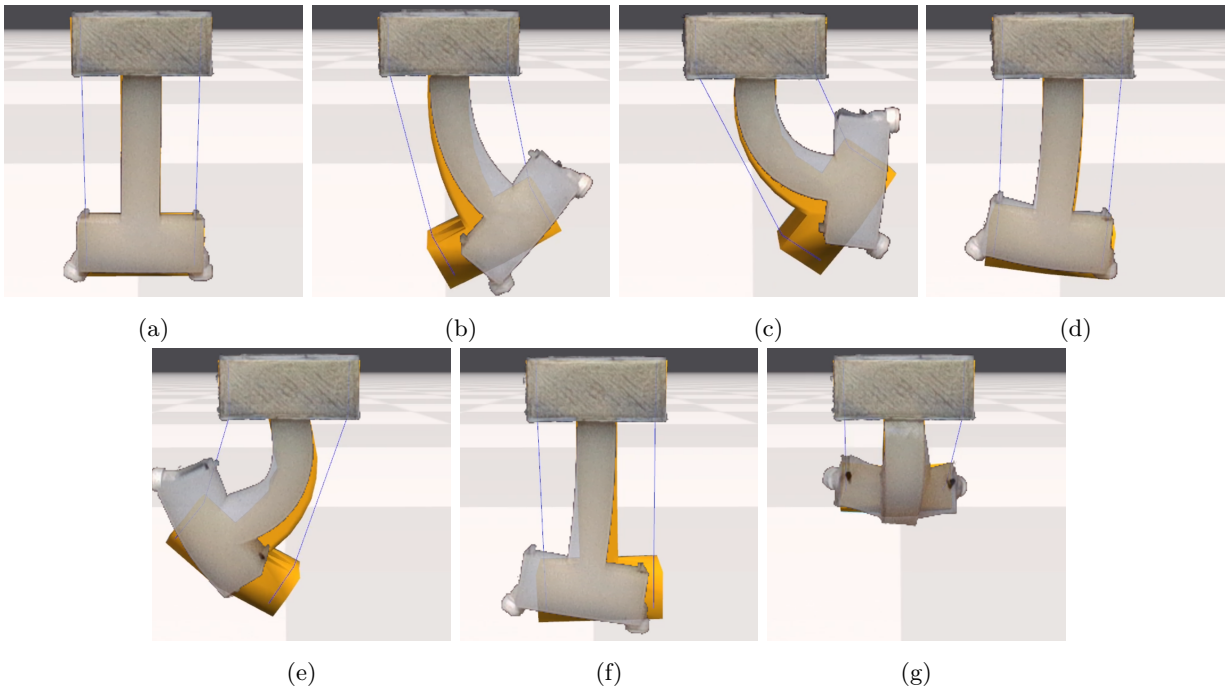


Figure 6.26: The simulated robot motion with physical robot motion overlaid.

From Figure 6.26 we note some key differences between the simulated robot motion and the motion of the physical robot. In Configurations (b), (c), and (e), it is clear that for the same control parameters, the deformations of the soft robot within our simulations are smaller than those we see in real life. Another important difference between the simulation and the real robot motion is that the soft body of the robot does not return to its resting position, even though the cables are reset to their initial length, as can be seen in Configuration (f). In other words, we experience a small amount of hysteresis, which might be caused by motor drag or the material of the cables (fishing wire). This unfortunately means that during a physical experiment the error introduced by hysteresis will keep on accumulating, and thus, the end result will drift away from our simulated motion. It is however interesting to note that the final configuration, (g), is very similar to the simulated configuration, even though the robot is affected by hysteresis.

The results from this experiment overall seem to indicate that our simulations do not capture the real world behaviour of the soft robots completely. The limitations of our simulated models seem to stem from four major error sources: the cable model not modelling friction between the cable and the soft body; hysteresis not being modelled by the simulator; the mesh of the robot not allowing for appropriate elastic behaviour; and the physical error sources such as motor drag, camera placement and the small fabrication inaccuracies.

6.4.2 Actuating the Spinebot

Introduction

In this experiment we actuate a physical copy of the 'Spinebot'. The goal of the experiment is to examine if the errors we observed in Experiment 6.4.1 are also present in our 'Spinebot' simulations.

Procedure

The physical robot and the rig we will be using to actuate our robot and capture motion data can be found in Figure 6.27.

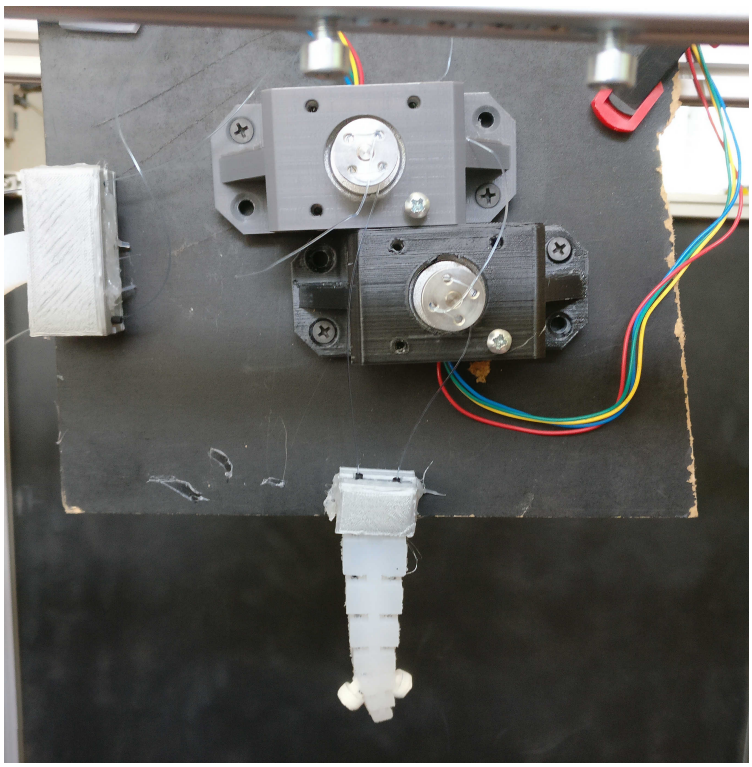


Figure 6.27: The physical set-up of the 'Spinebot' experiment. Notice that this is a dual rig, which has room for both the 'Hammerbot' and the 'Spinebot'.

The rest length of each of the cables - the part of the cables inside the robot - measure 80.5 mm, and in order to distribute forces at the cable end-points, we used small white plugs. We actuate the physical robot using the same configurations we applied to the robot in the simulations of Experiment 6.3.2. Observe in Table 6.5 the motor positions we will engage to reach the alpha values from Table 6.3. Both alpha values and motor positions have been added to the table, and each row corresponds to one configuration

Configuration	α_0 (Left cable)	α_1 (Right cable)	Motor 0	Motor 1
Configuration (a)	1.00	1.00	0	0
Configuration (b)	1.00	0.80	0	258
Configuration (c)	1.00	1.00	0	0
Configuration (d)	0.90	1.00	129	0
Configuration (e)	0.80	1.10	258	-129
Configuration (f)	1.00	1.00	0	0
Configuration (g)	0.90	0.90	129	129
Configuration (h)	0.80	0.80	258	258

Table 6.5: Table detailing the motor settings - in terms of steps - for each configuration we will examine.

As in Experiment 6.4.1 we will compare our findings to the simulated movement by comparing the configurations of the simulations to the real motion captured using an `Intel Realsense` camera.

Results

Observe in Figure 6.28 the robot actuated inside the Learning Cube.

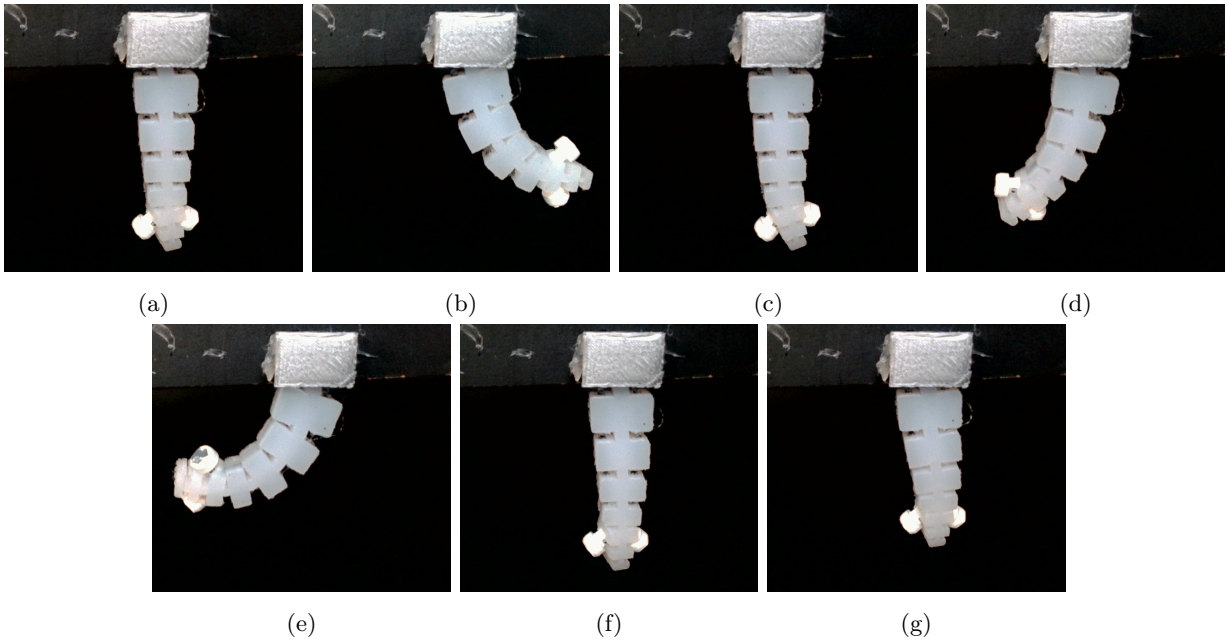


Figure 6.28: The 'Spinebot' during actuation. The part of the robot encased in gray plastic is being held in place to match the simulated set-up.

As is evident from Figure 6.28, the white plugs used to distribute forces at the end-point of the cable does not fit the robot very well. This causes it to slightly bend to the right in its resting position, as can be seen in Configurations (a) and (c). We can therefore not expect the physical and simulated robot motion to match perfectly, but we should still be able to compare the overall movement of the two. In Figure 6.29, the robot motion can be found, overlaid on the simulated results from Experiments 6.3.2.

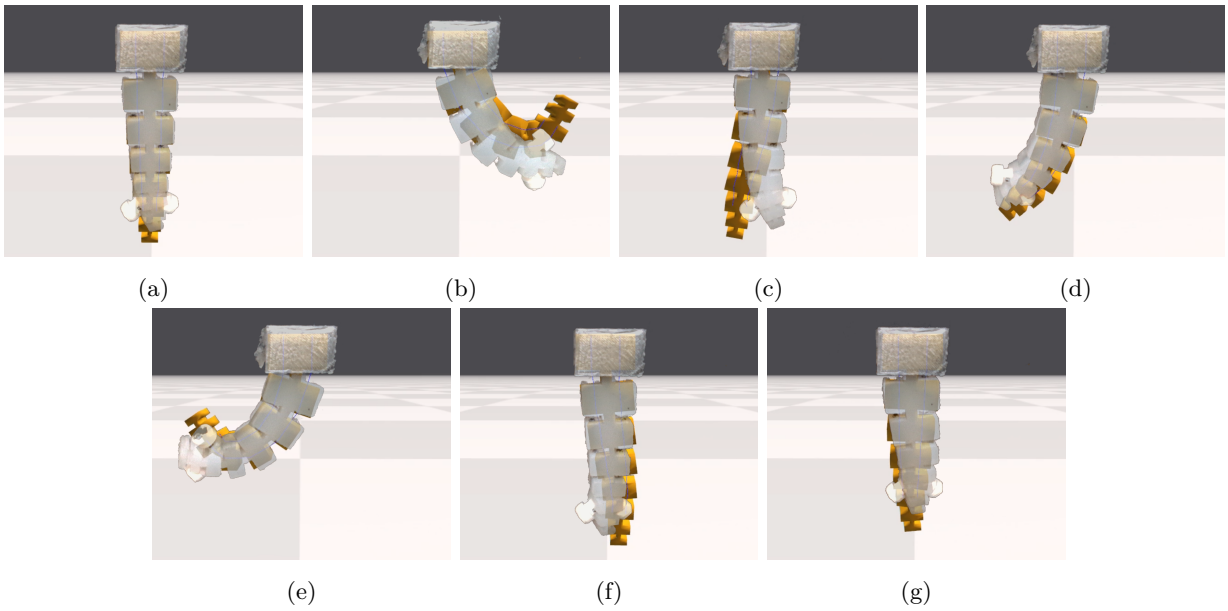


Figure 6.29: The simulated 'Spinebot' motion with the results of the physical experiment overlaid.

As in Experiment 6.4.1, we observe that our simulated motion does not match the true deformations of the physical robot. However, contrary to the 'Hammerbot', we actually see that the simulations 'overshoot' the real movement by producing larger deformations than we see in the real robot. One of the differences between the simulations of the 'Hammerbot' and the 'Spinebot' was that the 'Hammerbot' had considerably coarser mesh elements in the two 'T' heads. This might indicate that the mesh of the 'Hammerbot' made it act stiffer than one should expect from the material properties of the silicone. However, we also notice that in the physical robot we see that all joints of the robot body are activated during actuation. In the simulated version the tip of the robot is deformed significantly more than the rest of the body, which means that modelling cables as unilateral springs might not be sufficient for simulating true cable behaviour.

In terms of hysteresis, we see a similar trend in the 'Spinebot' - notice how Configurations (a), (c), and (f), in Figure 6.28 differ - which indicates that we need to either deal with hysteresis in the real world or model it in our simulations.

6.4.3 Actuating the Bubblebot robot

Introduction

In our pressure force experiments, we observed that the 'Bubblebot' did not behave as expected. However, we still need to compare the real robot motion to our simulated results, to confirm that our expectations were true.

Procedure

For this experiment we will be actuating all of the air-chambers of a physical copy of the 'Bubblebot' robot. Each chamber has an initial volume of 77.75 ml, and we will be using three syringes like the one from Figure 3.4 as actuation devices. The physical set-up of the experiment can be found in Figure 6.30

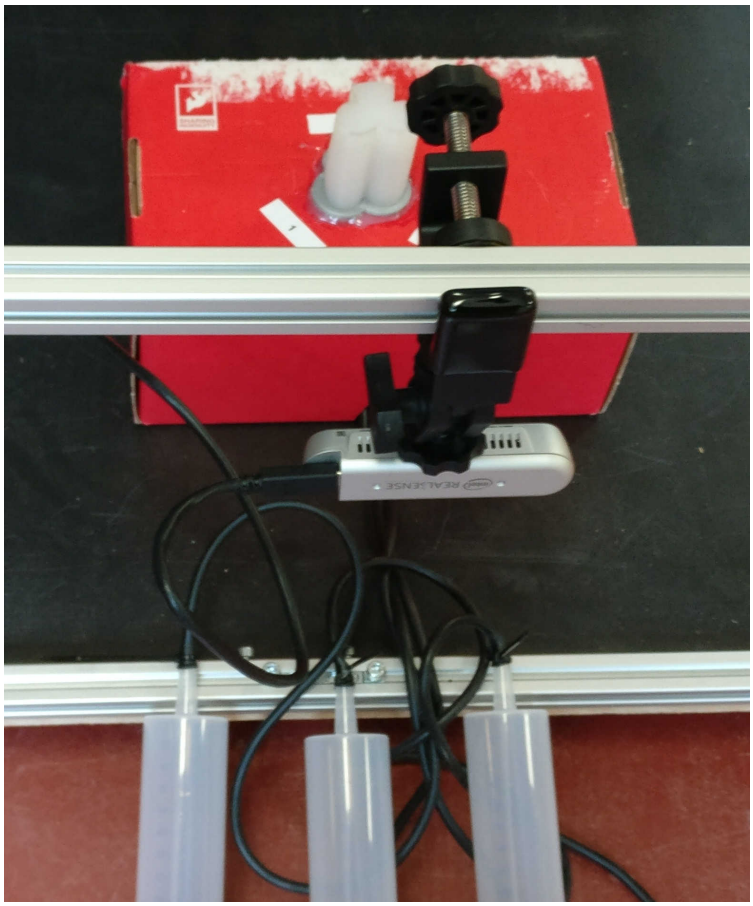


Figure 6.30: The physical set-up of the 'Bubblebot' experiment.

Notice that the camera is upside down, which will make the captured motion align with the simulations from Experiment 6.3.5. In Table 6.6 the amount of air we will use to inflate each chamber is given in ml.

Configuration	Pump 0	Pump 1	Pump 2
(a)	0	0	0
(b)	20	0	0
(c)	30	0	0
(d)	40	0	0
(e)	0	0	0
(f)	0	30	30
(g)	0	50	0
(h)	0	0	50
(i)	0	0	80

Table 6.6: Table detailing the pump settings - in terms of ml of air - for each configuration we will examine.

As the maximum factor that we will be able to increase the pressure with is $\frac{77.75ml}{77.75ml+110ml} = 2.4$ we will not be able to recreate the exact pressure from Experiment 6.3.5. Instead, we will compare the deformation with Configurations (b), (c), and (d), in order to determine how much

our simulations deviate from the real robot motion.

Results

Observe in Figure 6.31 the 'Bubblebot' robot during actuation.

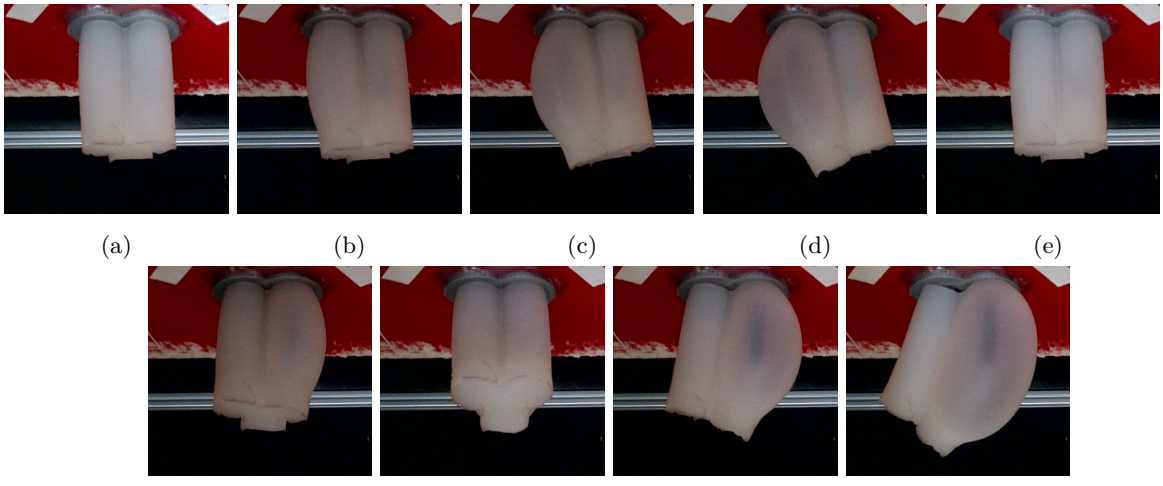


Figure 6.31: The 'Bubblebot' robot during actuation. The caption of each frame corresponds to a configuration from Table 6.6.

From Figure 6.31 we observe that the physical copy of the 'Bubblebot' robot requires a much lower pressure to achieve large deformations in its soft body compared to the simulation from Experiment 6.3.5. In Figure 6.32 the simulated robot - using the low quality mesh - can be found with the physical configurations (b), (c), and (d) overlaid.

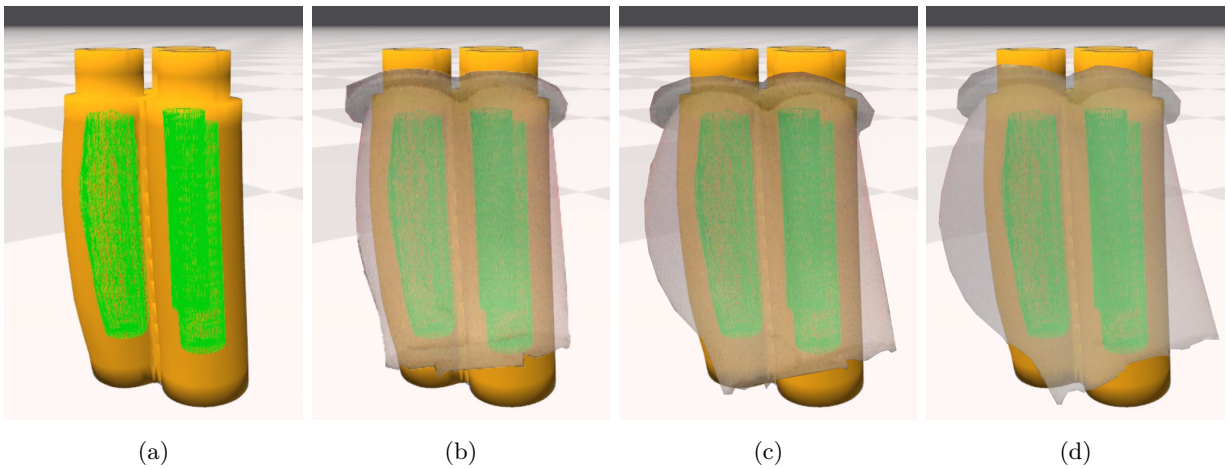


Figure 6.32: The simulation of the 'Bubblebot' robot with the physical robot Configurations (b), (c), and (d) overlaid.

These results indicate that the pressure forces experienced by the simulated model are either much lower than the ones experienced by the real robot, or that the volume of the air-chamber mesh is able to increase without causing the soft body of the robot to deform. A closer examination

of the air-chamber mesh of the simulated robot reveals that some of the triangles of the mesh seem to have inverted, which could be causing an erroneous volume estimation. In other words, these results support our previous discovery that we need a very high quality mesh to simulate accurate pressure forces.

6.5 Limitations of the Simulations

From the results of the experiments we have performed to investigate the gap between our simulations and reality, we have observed three major discrepancies between the real world and the simulated environment:

- The cable model does not capture the behaviour of a real cable.
- The simulator does not capture the hysteresis we observed in the physical experiments.
- Our meshes are of too low a quality for us to obtain meaningful pressure force simulations.

In our cable model, a cable is modelled as a number of unilateral springs connecting points embedded in a mesh. The reasoning behind the model is that with enough cable points, we can approximate the behaviour of a real cable well enough to produce realistic simulations of cables being pulled. However, in the Flex simulator we can only place cable points on the surface of our mesh, which means that we have almost no choice in terms of choosing the amount of cable points we want - in fact, we can have two at most for each place the cable cuts through the mesh. Furthermore, our cable is an implicit part of the simulation, meaning that we do not model the cable interacting with the soft body - i.e. collision, friction, etc. - and thus, we obviously cannot observe this behaviour in our simulations. In Figure 6.33 an example of how our cables can collide with a soft body can be found.

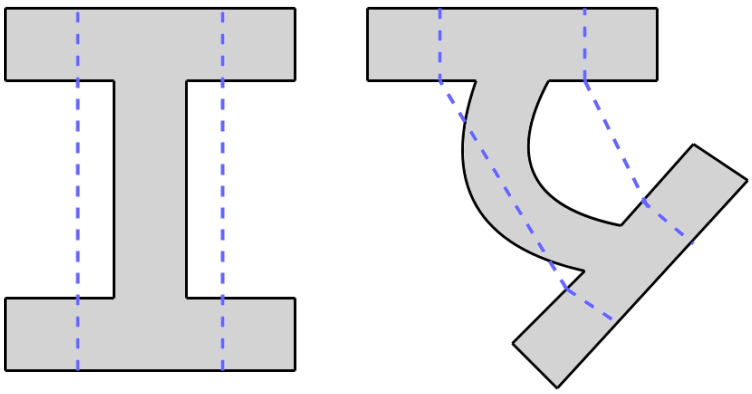


Figure 6.33: Example of a cable colliding with the soft body during actuation not being taken into account, since the cable is not being explicitly simulated.

In the simulated environment, our cables can only create forces near the cable points placed on the surface of the mesh, but in the real world colliding with a cable changes the deformation we observe significantly. This was evident in Experiment 6.4.1 in Figure 6.25e, where the bending part of the robot was caught on a cable similarly to the example from Figure 6.33.

This limits the realism of our robot in terms of relatively large deformations, but due to the small amount of cable points we can place on the mesh, the fidelity of smaller movements are also

lost. Observe in Figure 6.34 a physical copy of the 'Spinebot' next to a simulated model, where we can see how the smaller parts of the robot - near the tip - behaves quite differently in real life and in the simulated environment.

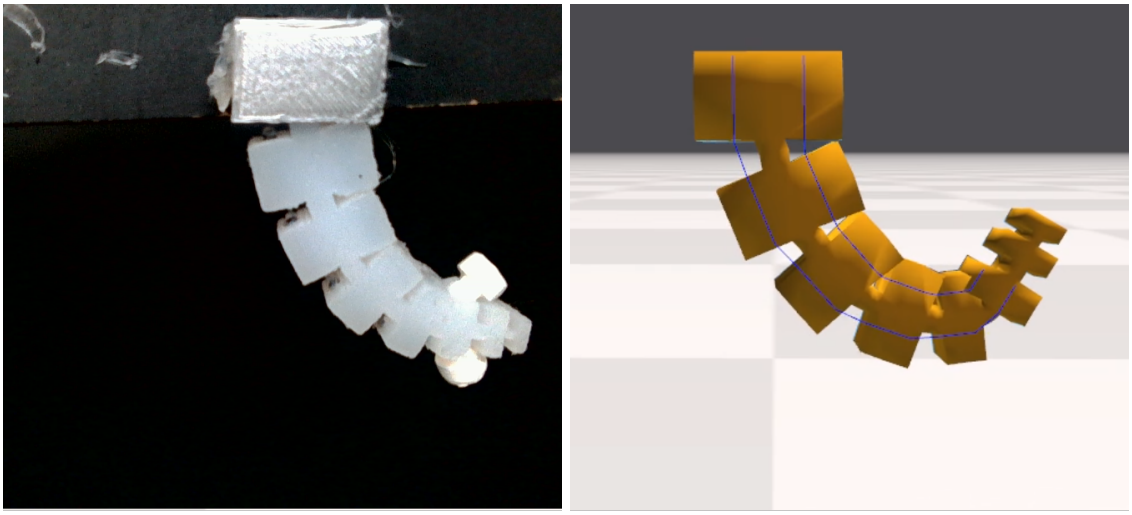


Figure 6.34: Example of how modelling a cable using springs means that friction forces are not modelled within the soft body. The simulated robot can be seen as performing a series of smaller motions rather than the single smooth movement we see in the real world.

Ideally we would want to model our cables as explicit parts of the simulation, such that we can simulate friction and contact forces between the cables and the soft body, but using enough cable points we should still be able to observe something, that at least visually approximates real world behaviour. However, the too coarse discretisation of our cables lead to unrealistic cable forces, and therefore, low fidelity simulations.

The cable forces that we produce during actuation are difficult to distribute correctly across the elements of the mesh, and in our experience small elements tend to compress under the forces produced by the cables being pulled. To preserve the behaviour of the soft body near cable points, we have had to increase the size of the tetrahedra elements near these points, since this creates an artificial stiffening of the material due to numerical stiffnening [Che+17]. In Figure 6.35 the mesh of the 'Hammerbot' can be found. Notice that the size of the elements are greatly increased near the 'T' heads of the robot, while the beam connecting the heads consists of much smaller elements.

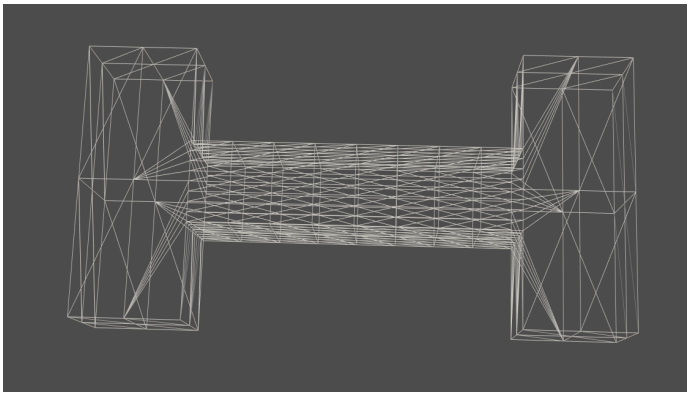


Figure 6.35: The mesh of the 'Hammerbot'. Notice the large discrepancies between mesh element sizes.

In our physical experiments we observed that our robots did not return to their resting positions when the cables where brought back to their original rest lengths. In other words, we observe a form of hysteresis in the soft bodies. Hysteresis depends on several factors such as how well the silicone rubber is mixed, the quality of the degassing process, and even depends on the room temperature during the experiment [Rey+13]. In [BKC17] hysteresis was dealt with by changing the properties of the soft robot - i.e. increasing the stuffing in their plush robots - causing the internal forces of the robot to work as an antagonistic actuator. Due to the simplicity of the manufacturing process, the problem of our robots does not seem to be caused during the fabrication process - since the pressure-driven robot did not experience the same hysteresis issues - but instead by the lack of antagonistic actuation in our cable-driven robots. For all pressure-driven silicone robots, the internal elastic forces of the silicone will always oppose any actuation that aim to expand the volume, which gives us the antagonistic relationship we are looking for in a soft robot. To emulate antagonistic actuation in our cable-driven experiments we placed the robots upside-down, since gravity would then act as an antagonistic force - however, the gravitational forces were clearly not sufficient to properly reset the robot.

Now, even though the pneumatically actuated robots exhibits preferable behaviour in terms of re-entering its resting state, our simulations using pressure forces have proven to be rather volatile. In our cable force simulations, only the elastic response of the soft body of the robot depended on the quality of the mesh, but for our pressure force computations, a low quality mesh quickly compounds errors as a large part of the computation involves using face normals and areas, as well as the volume of the air-chambers. Therefore, our pressure-driven robot simulations are a lot less robust than the cable-driven robot simulations. In conclusion, our pressure force model seem to be too unstable with our current level of mesh quality to be used for learning control policies, and some air-chamber designs have produced simulations too inconsistent with the real world to even allow for an exploration of the workspace of the robot to be possible. Our cable-driven robots on the other hand, have proven to be a lot more reliable. We have observed a gap between our simulations and reality, but unlike our pressure force simulations, actuating the cable-driven robots in a simulation provides a fairly good idea about how the robot will behave in the real world too.

Chapter 7

Control Policy Optimisation

To some extent we can now simulate robot motion that approximates real world behaviour, but so far we have yet to define an objective that we want the robots to complete. When thinking about how our robots can become useful, tasks such as grasping and moving objects, exploring dangerous areas, and assisting doctors during surgery, are some of the tasks that comes to mind. However, to stay within the scope of this thesis, we will only concern ourselves with much simpler tasks that can be defined as minimising a smooth and strictly convex function with respects to the control parameters of the robots. More formally, for a smooth and strictly convex objective function, $\mathcal{O}(\mathbf{x}(\alpha))$, we want to find the optimal control parameters, $\alpha^\star$, given by,

$$\alpha^\star = \underset{\alpha}{\operatorname{argmin}}\, \mathcal{O}(\mathbf{x}(\alpha)) \tag{7.1}$$

where $\mathbf{x}(\alpha)$ is the robot configuration resulting from actuating the robot according to control parameters α. Smooth and strictly convex functions are easy to work with, as a solution - i.e. a local minimiser - to the objective function, will always be the optimal control parameters in terms of solving the task [NW06a]. In other words, we know that any $\alpha^\star$ that minimises $\mathcal{O}(\mathbf{x}(\alpha))$ is the global minimiser of the objective function. This way of finding $\alpha^\star$ of course means that we ignore the dynamics of our robots behaviour, and instead treat the problem as if it was quasi-static.

In this chapter, we will present two different approaches to solve an optimisation problem of the form given by Equation (7.1). We will then explore how well these learned policies transfer to the real world, and finally discuss the state-of-the-art in learning to control robots, with a focus on transferring policies from simulations to reality. Due to our pressure-driven simulations having proven to be unreliable, and the physical actuators lacking high-precision control, we will only concern ourselves with learning control policies for cable-driven soft robots.

7.1 Steepest Descent

The first method we will use to minimise our objective is *steepest descent*. Steepest descent is a line search strategy, which searches for a minimiser in the direction of steepest descent - hence the name - given by $\nabla_\alpha \mathcal{O}(\mathbf{x}(\alpha))$, as this is the direction in which $\mathcal{O}(\mathbf{x}(\alpha))$ changes most rapidly [NW06b]. An advantage of steepest descent is that it only requires the calculation of the gradient, unlike many other optimisation schemes, where the second order derivative is also commonly required. As steepest descent is a line search algorithm, its iteration is given by,

$$\alpha^{(t+1)} = \alpha^{(t)} - \eta^{(t)} \nabla_\alpha \mathcal{O}(\mathbf{x}(\alpha)) \tag{7.2}$$

where $\eta^{(t)}$ is the step length of the line search. Choosing $\eta^{(t)}$ is important in terms of ensuring convergence of the line search method, and in this thesis we will be using *Root-Mean-Squared Propagation* (RMSprop) of the gradient to choose a suitable step length in each iteration. RMSprop chooses $\eta^{(t)}$ based on the previous behaviour of the gradient, which means that when we are close to the minimiser, we are still able to take steps that are large enough to ensure a significant decrease in function value. Using RMSprop, the steepest descent iteration is now given by,

$$\begin{aligned} S^{(t)} &= \gamma S^{(t-1)} + (1-\gamma)\Big(\nabla_\alpha \mathcal{O}(\mathbf{x}(\alpha))\Big)^2 \\ \alpha^{(t+1)} &= \alpha^{(t)} - \frac{\eta}{\sqrt{S^{(t)}+\varepsilon}}\nabla_\alpha \mathcal{O}(\mathbf{x}(\alpha)) \end{aligned} \tag{7.3}$$

where $S^{(0)} = 0$, $\gamma \in (0,1)$ is a discounting factor, $\eta > 0$ is called the learning rate and $0 < \varepsilon \ll 1$ is a small number that ensures that we do not divide by 0. In our experience, setting $\gamma = 0.9$, $\varepsilon = 10^{-6}$ and $\eta = 0.001$, has shown satisfactory results in terms of convergence rate for our purposes.

As part of the steepest descent algorithm we need to compute the derivative of $\mathcal{O}(\mathbf{x}(\alpha))$, but it might not always be possible to do this analytically. The objective function that we will be using for our experiments in this thesis is given by,

$$\mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) = \frac{1}{2}(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))^T(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)})) \tag{7.4}$$

where $\mathbf{y}$ is some target position, and $\mathbf{x}_e(\alpha^{(t)})$ denotes the position of the point e on the robot resulting from actuation according to the parameter $\alpha^{(t)}$. Here, the gradient of the objective function using a robot with n control parameters is given by,

$$\nabla_\alpha \mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) = \begin{bmatrix} \frac{\partial \mathcal{O}(\mathbf{x}_e(\alpha^{(t)})}{\partial \alpha_0} \\ \vdots \\ \frac{\partial \mathcal{O}(\mathbf{x}_e(\alpha^{(t)})}{\partial \alpha_{n-1}} \end{bmatrix} = \begin{bmatrix} (\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))\frac{\partial}{\partial \alpha_0}\mathbf{x}_e(\alpha^{(t)}) \\ \vdots \\ (\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))\frac{\partial}{\partial \alpha_n}\mathbf{x}_e(\alpha^{(t)}) \end{bmatrix} \tag{7.5}$$

but since we do not model $\mathbf{x}_e(\alpha^{(t)})$ as a differentiable function, we can only approximate the partial derivative $\frac{\partial}{\partial \alpha_i}\mathbf{x}_e(\alpha^{(t)})$. To perform this approximation, we will use a finite difference approximation as described in [NW06c]. A common way to perform the finite difference approximation is to use a *forward-difference* scheme as,

$$\frac{\partial f}{\partial x}(x) \approx \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{7.6}$$

as this resembles the true derivative given by,

$$\frac{\partial f}{\partial x}(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{7.7}$$

However, for our cable-driven robots, choosing a forward difference approach - given that all control parameters are set as $\alpha_i = 1, \forall i$ at the beginning of the simulation - is a bad idea, as we have,

$$\mathbf{x}_e(\alpha^{(t)} + \Delta\alpha) = \mathbf{x}_e(\alpha^{(t)}) \tag{7.8}$$

due to the fact that we allow our cables to slack. We will therefore be using a *backwards-difference* approximation given by,

$$\frac{\partial \mathbf{x}_e}{\partial \alpha_i}(\alpha^{(t)}) \approx \frac{\mathbf{x}_e(\alpha) - \mathbf{x}_e(\alpha - \Delta\alpha_i)}{\|\Delta\alpha_i\|} \tag{7.9}$$

where $\Delta\alpha_i$ is a vector with 0 values for all $j \neq i$ in α.

7.2 PID Controller

The standard approach to control system behaviour has for a long time been to use a *Proportional-Integral-Derivative* (PID) scheme. As the name suggests, a PID controller works by combining the proportional error - i.e. the deviation from the target state - with the integral of the error and the derivative of the error [Åst02], and outputs a control signal $u(t)$ that aims to minimise the error function. The control signal is given by,

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{\partial}{\partial t} e(t) \tag{7.10}$$

where, K_p, K_i, K_d are the proportional gain, integral gain, and derivative gain, respectively, and the error term $e(t)$ is found by evaluating the objective function at time t.

The main advantage of a PID controller lies in the usage of the integral and derivative terms. The integral term makes sure that the magnitude of the error works in the favour of the minimisation by making sure that if the optimum - in terms of control signal - is overshot, a correspondingly large step is taken in the opposite direction during the next iteration. The addition of the derivative term then allows the control signal to peek into the future in terms of how the objective function behaves, and can be seen as always taken a small step in the direction of steepest descent. For the objective function given by,

$$\mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) = \frac{1}{2}(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))^T(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)})) \tag{7.11}$$

the control signal at time t for the k'th control parameter is computed as,

$$u_k(t) = K_p\mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) + K_i \int_0^t \mathcal{O}(\mathbf{x}_e(\alpha^{(\tau)}))d\tau + K_d(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))\frac{\partial}{\partial \alpha_k}\mathbf{x}_e(\alpha^{(t)}) \tag{7.12}$$

which means that we need to compute both the partial derivative of $\mathbf{x}_e(\alpha^{(t)})$ and the definite integral of $\mathcal{O}(\mathbf{x}_e(\alpha^{(t)}))$. We will again use a backwards-difference approximation to compute the partial derivatives, and to approximate the definite integral we will use Simpson's rule as stated in [Atk89]. Thus, we will approximate the integral as,

$$\int_0^t \mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) \approx \frac{t}{6}\Big(\mathcal{O}(\mathbf{x}_e(\alpha^{(0)})) + 4\mathcal{O}(\mathbf{x}_e(\alpha^{(t/2)})) + \mathcal{O}(\mathbf{x}_e(\alpha^{t}))\Big) \tag{7.13}$$

which means that the 'approximated' control signal, $u_k^{\sim}(t)$ is given by,

$$\begin{aligned} u_k^{\sim}(t) &= K_p\mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) \\ &+ K_i\frac{t}{6}\Big(\mathcal{O}(\mathbf{x}_e(\alpha^{(0)})) + 4\mathcal{O}(\mathbf{x}_e(\alpha^{(t/2)})) + \mathcal{O}(\mathbf{x}_e(\alpha^{t}))\Big) \\ &+ K_d(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))\frac{\mathbf{x}_e(\alpha) - \mathbf{x}_e(\alpha - \Delta\alpha_k)}{\|\Delta\alpha_k\|} \end{aligned} \tag{7.14}$$

This control signal expresses a change in the k'th control parameter, which means that we will use the PID control signal to control our cable-driven robots as,

$$\alpha_k^{(t)} = u_k^{\sim}(t) + 1 \tag{7.15}$$

since we assume that $\alpha_k^{(0)} = 1, \forall k$.

We want to choose our gain parameters, K_p, K_i, K_d, such that we quickly converge towards the optimal control parameters, but without creating large oscillations in the control signals by overshooting the target. We achieve this by experimentally finding suitable parameters, but we recognise that it would probably be possible to achieve better results by performing a more sophisticated parameter tuning. The values we have found to exhibit the expected behaviour of the PID controller was,

$$\begin{aligned} K_p &= 0.05 \\ K_i &= 0.00075 \\ K_d &= 150 \end{aligned} \tag{7.16}$$

which we will use in all experiments involving the PID controller.

7.3 Experiments

7.3.1 Inverse Kinematics using Steepest Descent

Introduction

As described previously, we model the inverse kinematics problem as,

$$\mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) = \frac{1}{2}(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))^T(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))$$

and in this experiment we want to examine if we are able to solve the problem using a steepest descent approach.

Procedure

In this experiment we will actuate the 'Hammerbot', in order to minimise the distance between a point placed near one end of the robot and a target position. Observe in Figure 7.1 the robot with its endpoint visualised as a green cross.

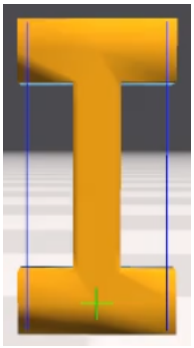


Figure 7.1: The robot with endpoint visualised as a green cross.

We will choose 10 random points near the lower half of the robot, with a minimum distance of 25 mm to the centre of the robot. The z-coordinate of each point will be fixed, such that the variation in position will only be in the xy-plane. This is the plane where points can be reached by actuating one cable at a time, while changes to the z-coordinate of the endpoint requires the

cables to be actuated at the same time. However, as we only examine the partial derivatives approximated from actuating cables individually, we cannot learn to produce this behaviour.

For each randomly chosen point, we will run the line search algorithm until we see a gradient magnitude of less than 10^{-4} for both cables, or the line search has performed more than 1,000 iterations. We will present the convergence of the objective function and perform a visual inspection of the robots behaviour.

Results

Observe in Figure 7.2 the points that were randomly chosen, and in Figure 7.3 the convergence of the objective function as a function of iterations for all of the target positions.

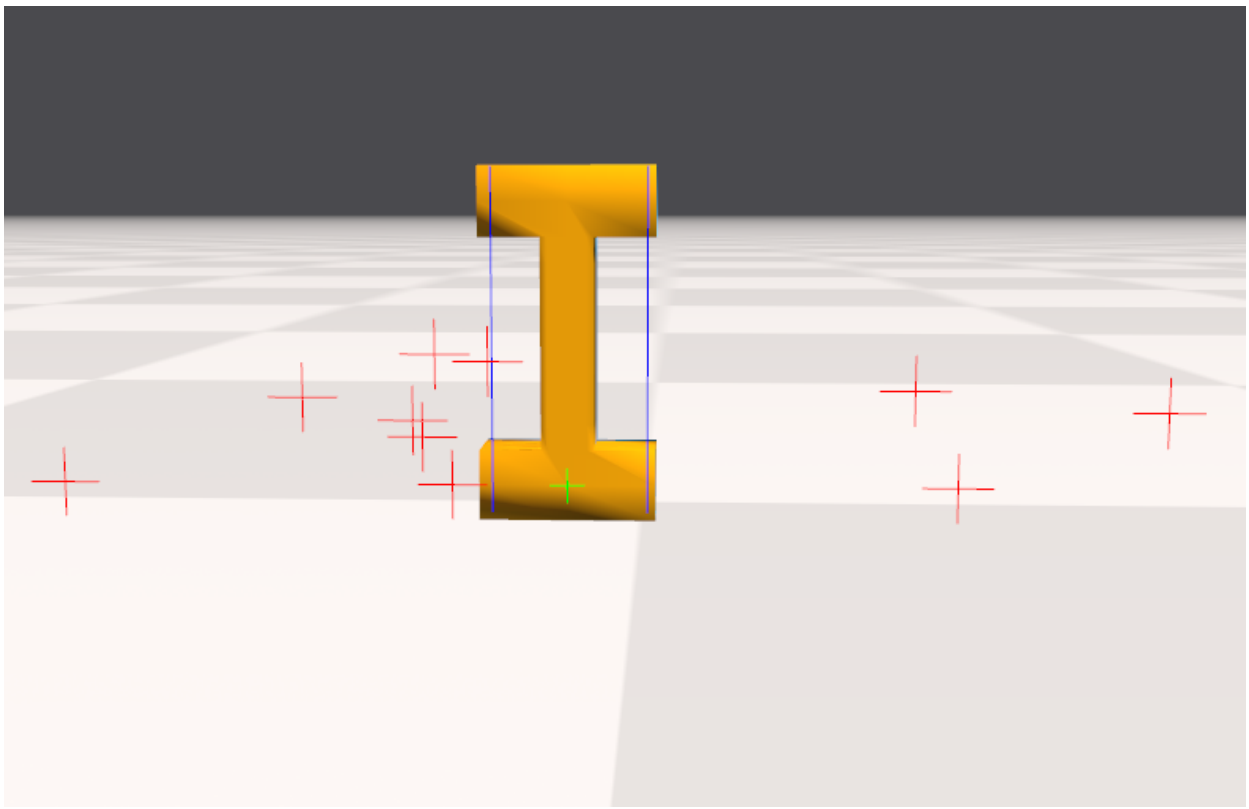


Figure 7.2: The 10 randomly chosen points in red.

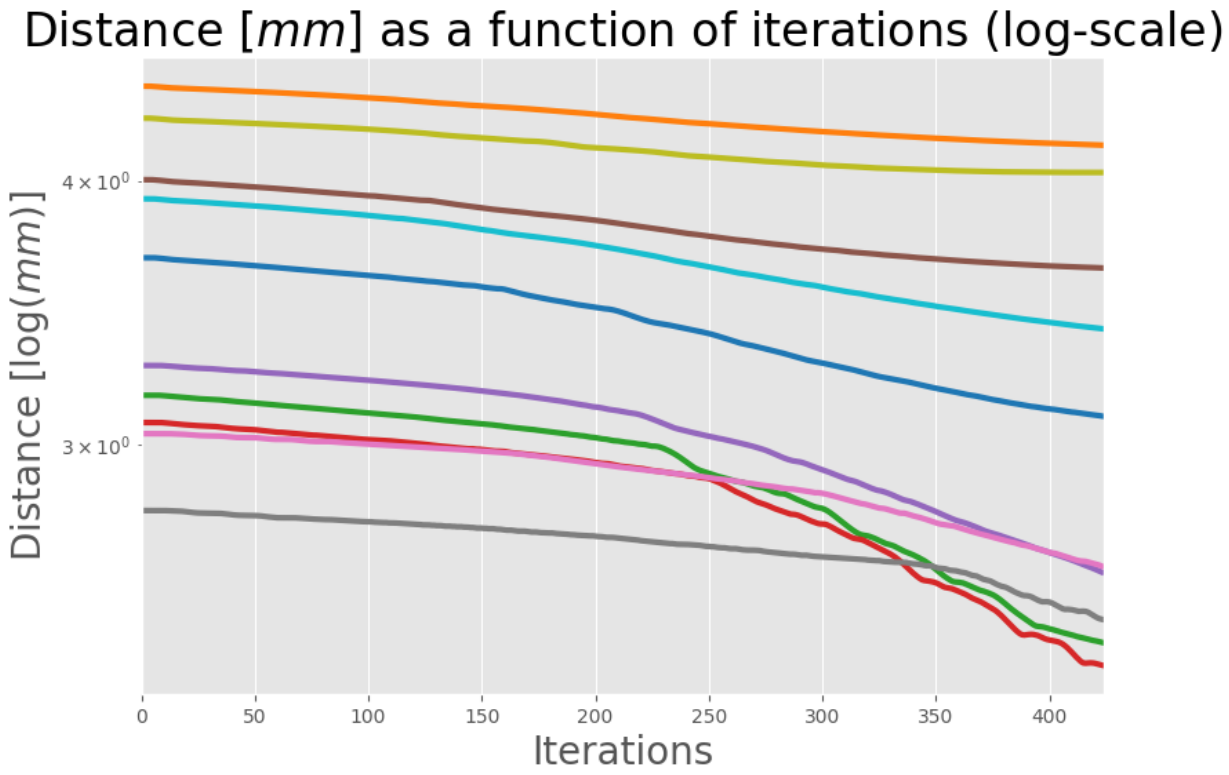
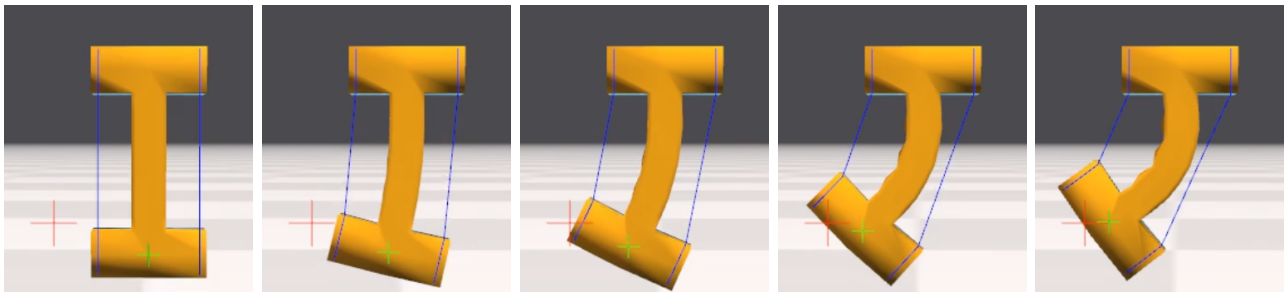


Figure 7.3: Convergence plot of the steepest descent line search. Notice that the y-axis - i.e. distance between target and endpoint - is in logarithmic scale.

Here, we observe that the objective function in general does not converge linearly, as we would have expected of the steepest descent algorithm. Instead we observe something that resembles a mixture of sub-linear and linear convergence. We expect this to be caused by the gradient being approximated, as we therefore cannot be sure that we take a step in the exact direction of steepest descent in each iteration. A visual examination of the optimal control scheme derived from one of the minimisations can be found in Figure 7.4, which shows still shots of the robot from beginning to the end of simulation.



(a) (b) (c) (d) (e)

Figure 7.4: The 'Hammerbot' minimising the distance between its endpoint (green) and one of the randomly chosen points (red).

From these results we notice that the steepest descent algorithm seems to be well suited for

minimising the objective function. However, as we observed in our experiments from Chapter 6, the deformation of the robot depends on the order in which the cables were actuated. In that sense, we have only presented the results of 'easy' cases - i.e. confining the target positions to the xy-plan - but the results are very promising in terms of learning to control our robots.

7.3.2 Inverse Kinematics using the PID Controller

Introduction

In this experiment, we want to examine if we can learn to solve the inverse kinematics task using a PID controller.

Procedure

For this experiment we will use the PID controller given by Equation (7.14) to minimise the objective function,

$$\mathcal{O}(\mathbf{x}_e(\alpha^{(t)})) = \frac{1}{2}(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))^T(\mathbf{y} - \mathbf{x}_e(\alpha^{(t)}))$$

We want to compare the performance of the PID controller to the steepest descent algorithm, so we will therefore be performing the experiments on the same robot, using the same target positions. We let the PID controller perform 100 iterations, and expect to see a result similar to the steepest descent approach.

Results

Observe in Figure 7.5 the trend of the minimisation of the objective for the 10 different target positions.

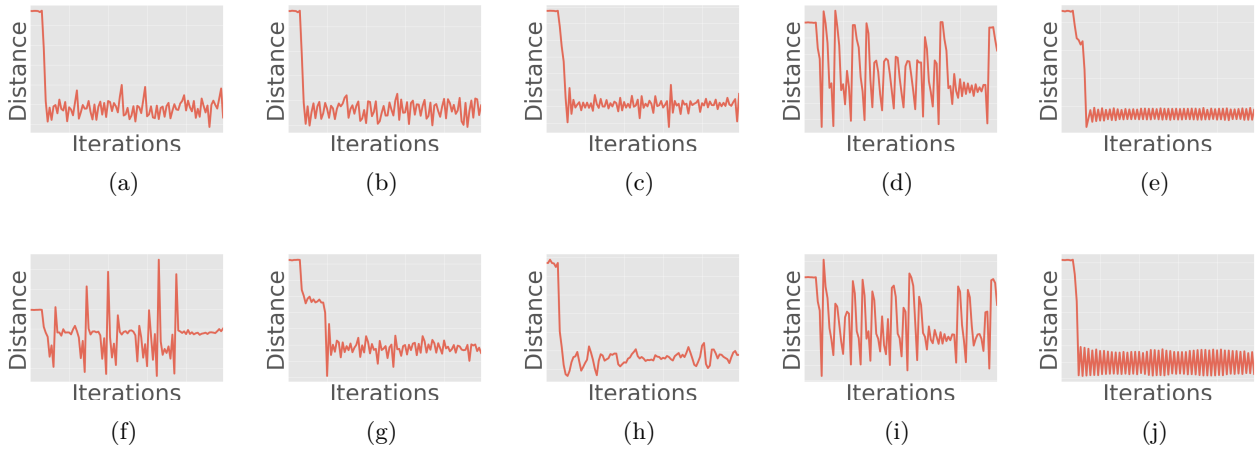
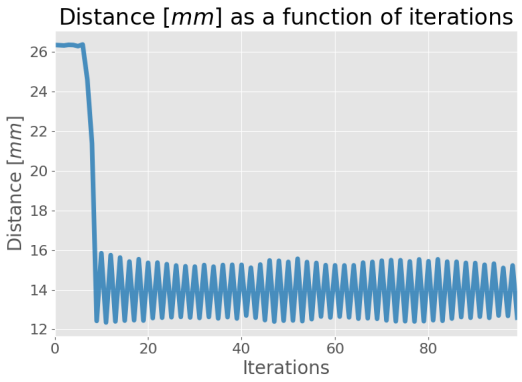


Figure 7.5: The distance to the target position as a function of iterations performed by the PID controller.

From these results we notice that for most of the experiments, the PID controller behaves as expected - i.e. quickly reaches the minimiser and then oscillates slightly - but for some of the experiments, we observe very large oscillations even at the end of the simulation. In Figure 7.6 the plots from Figures 7.5d and 7.5j can be found, containing additional details.

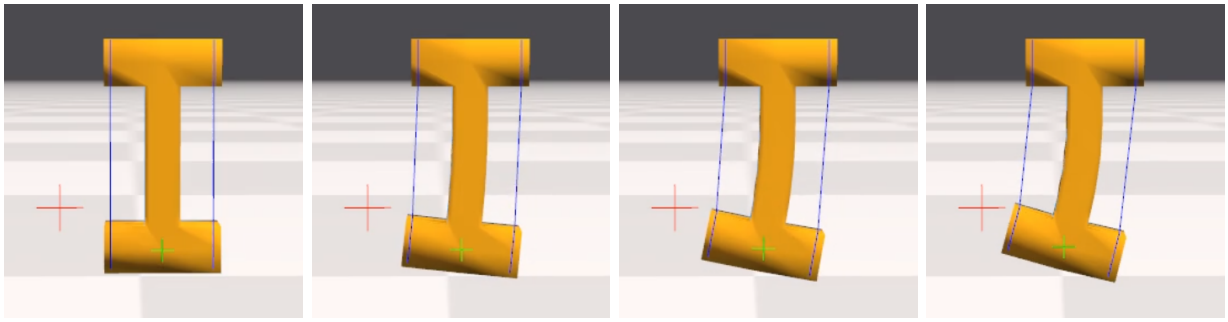
(a) An example of big oscillations, making the output signal unreliable.



(b) An example of the PID controller output signal converging quickly to the minimiser of the objective function.

Figure 7.6: The best and the worst experimental results in terms of the behaviour of the objective function.

From this comparison we quickly notice that the PID controller behaves unexpectedly in the plot to the left. However, a closer examination of the plots reveal that the initial distance between the target points and the endpoint of the robot varies greatly. In the 'good example' (Figure 7.6b), the distance to the target starts at $\sim$ 26 mm, while the initial distance in the second plot is significantly higher, at $\sim$ 75.5 mm. This seems to indicate that the proportional gain of the PID controller causes the controller to behave too violently due to the large distance to the target position. Performing a visual inspection of the actuation resulting from the minimisation of the objective function can be seen in Figure 7.7



(a) (b) (c) (d)

Figure 7.7: The robot being actuated at the beginning and end of the simulation at roughly the same intervals.

From these results, it is clear that the control signals, $u_k^{\sim}(t)$, do not express the true control parameters needed to minimise the objective function. Comparing the PID controller and the steepest descent algorithm in terms of distance to the target at the end of simulation shows that this is the case for all target points, as can be seen in Figure 7.8.

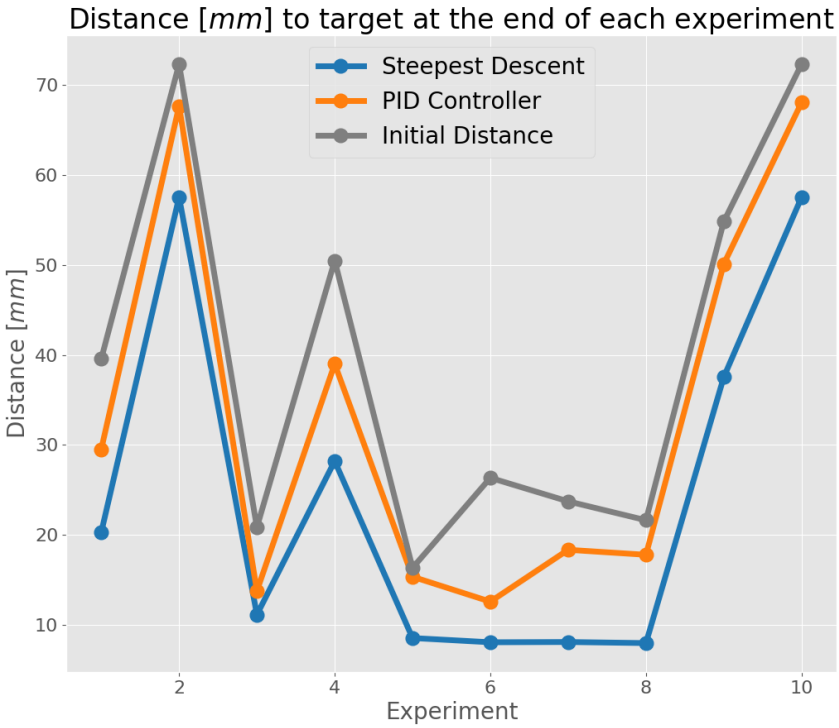


Figure 7.8: A comparison between the steepest descent algorithm and the PID controller in terms of the distance to target position at the end of the simulation. The blue circles mark the distance to the target points at the end of the steepest descent approach, while the orange circles mark the distance for the PID controller. Finally, the gray circles are the distance at the beginning of the simulation.

Here, we notice that the PID controller performs approximately half as well as the steepest descent scheme, which indicates that the parameters of the controller might not have been tuned well enough. A PID controller is mostly used when we have a lot of prior knowledge about the system we wish to control, which includes the objective. However, for all of our experiments, we have used randomly selected target positions, which means that we have not been able to tune the PID controller to perform any task perfectly. These results seem to suggest that the steepest descent approach is better suited for minimising objective functions which include a measure of randomness, due to how we can adapt the step length parameter of the line search dynamically.

7.3.3 Transfer of Inverse Kinematics Control Policy

Introduction

In this experiment we want to examine how well we can transfer a policy - learned using only simulated data - to control a real soft robot.

Procedure

In this experiment we will be actuating the 'Spinebot' and reduce the distance between the tip of the robot - indicated by a red marker - and a point in space. In Figure 7.9 the robot can be found, next to the point in space - i.e. the centre of a white ball - where we wish to move the red marker.

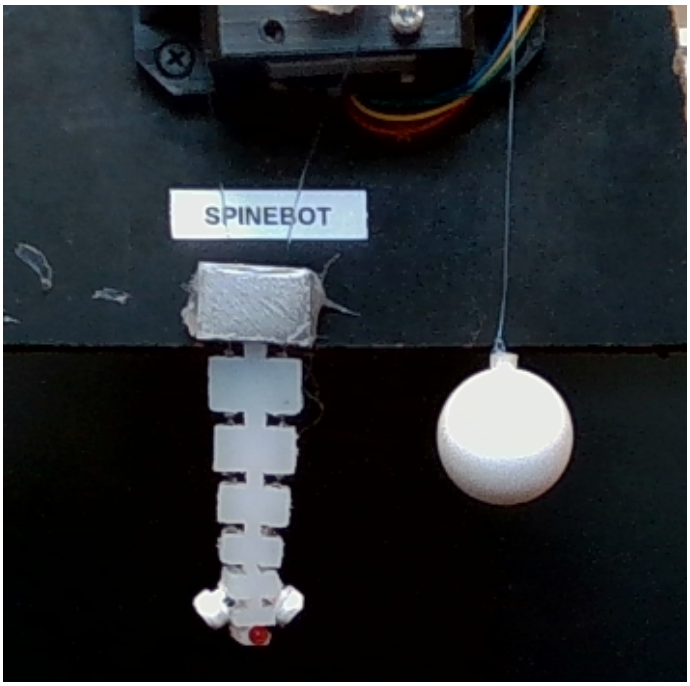


Figure 7.9: The robot with a red marker placed, next to the white ball we want to 'touch'.

The ball was placed exactly 60 mm from the centre of the robot, which means that the target point should lie within the work space of the robot. In Figure 7.10 the simulated set-up of the robot can be found. This is the set-up that will be used to learn the control policy, which we will transfer directly to the real robot.

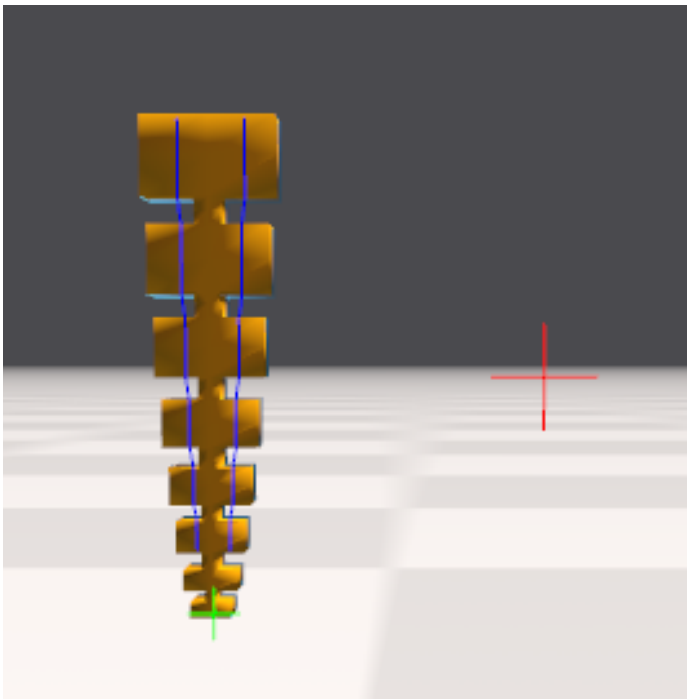


Figure 7.10: The simulated robot we will use to learn our control policy. Here, the green cross indicates the position of the red marker on the real robot, and the red cross indicates the centre of the white ball.

We train the simulated model using the steepest descent approach, and let the line search iterate until the gradient magnitude is smaller than 10^{-6} or more than 1000 iterations have been taken.

Results

Observe in Figure 7.11 the convergence of the steepest descent algorithm minimising the distance between the marker - the green cross - and the centre of the white ball - the red marker.

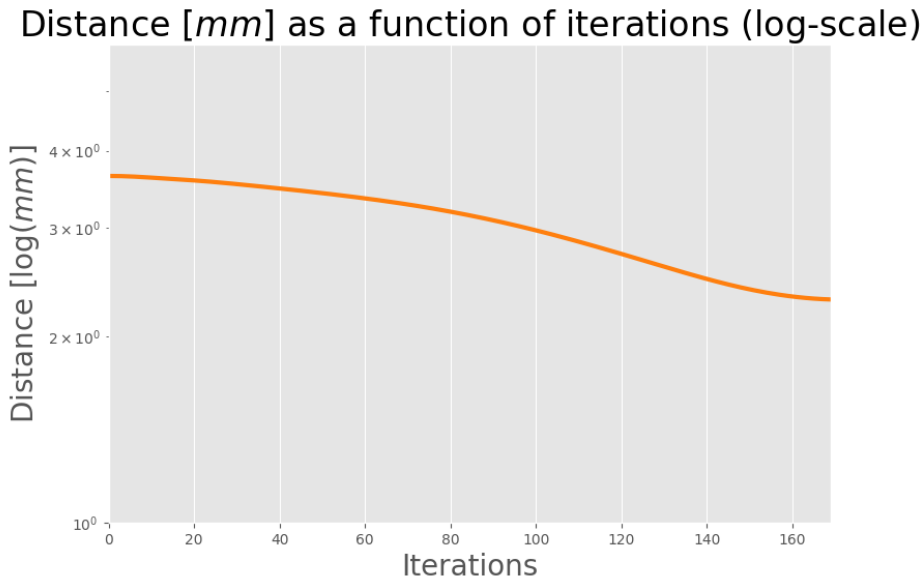
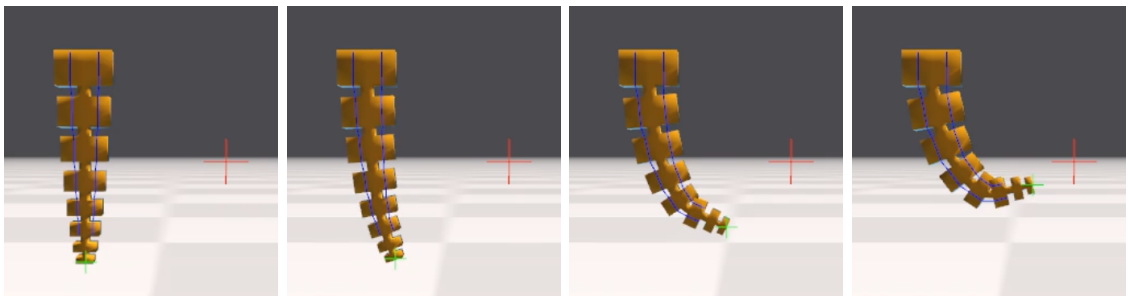


Figure 7.11: The convergence of the objective function for the simulated set-up from Figure 7.10.

As for the steepest descent experiments using the 'Hammerbot', we notice that the algorithm again converges somewhat linearly. Using the control parameters at the end of the minimisation resulted in a reduction in the distance between the target and the robot from 38.15 mm to 13.95 mm. In other words, the robot was not successful in terms of 'touching' the point. This can also be observed in Figure 7.12, where still shots of the simulated robot during actuation can be found.

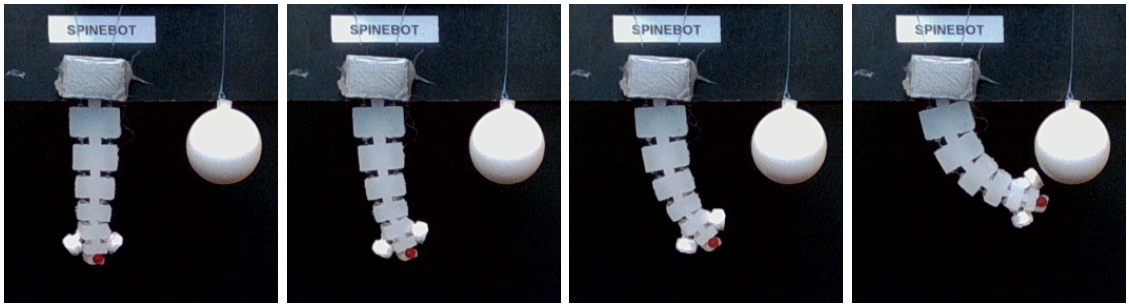


(a) (b) (c) (d)

Figure 7.12: The simulated robot being actuated at the beginning and end of the simulation at roughly same sized intervals, using the learned policy.

The robot exhibits the correct tendency in its behaviour - i.e. points towards the target - but

due to issues with the cable model discussed earlier, we observe that the robot self-intersect near the tip, resulting in the robot being unable to reduce the distance to the target any further. In Figure 7.13 the physical copy of the 'Spinebot' can be found while being actuated using the control policy learned from the simulation.



(a) (b) (c) (d)

Figure 7.13: The physical robot being actuated according to the control policy learned through simulated training.

In the physical robot we observe some discrepancies from the simulated motion, but overall the control policy seems to transfer well, considering the issues we have observed in the cable model. However, as long as the simulations contain errors such as robots self-intersecting, it is futile to assume that a policy can transfer directly from the simulated environment to the real world.

7.4 State-of-the-Art in Learning Robot Control

In recent years, state of the art in solving complex tasks using robots seem to have shifted towards using *reinforcement learning*. In general, a task can be defined as difficult whenever it is hard to define an objective function, which when minimised can claim to have solved the task. As we have seen, solving for inverse kinematics would not be classified as 'difficult' under this definition, as we can easily model a well-behaved objective function. When we move on to tasks where we want the robot to interact dynamically with its environment, it becomes less clear how exactly our objective functions is supposed to capture entities out of our control, such as other robots or static immovable objects.

A reinforcement learning (RL) algorithm learns to solve a task by interacting directly with the environment, while exploring the nuances that we find difficult to describe as a function. More formally, the RL algorithm uses a *learning agent* - in the case of robotics this is often a model of the robot itself - to learn a mapping from the *state space* of our environment, to the *action space* of our robot [SB18]. In the case of robotics, the state space is often determined by the internal state of the robot combined with an approximate state of the environment, which is typically acquired using some vision-based system. The goal of the learning agent is to learn the most favourable mapping from states to actions in terms of completing a given task. We therefore need to specify a *reward function*, which provides a feedback to the agent determining whether an action was good or bad. Using the feedback, the agent develops a *policy* by reinforcing positive behaviour and discouraging actions that negatively impact performance. Therefore, the key to ensuring that the robot learns to exhibit the desired behaviour lies in the design of the reward function. Observe

in Figure 7.14 a flow-diagram detailing how the agent and environment interact in the basic RL setting.

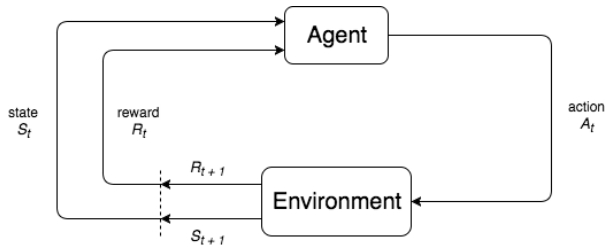


Figure 7.14: An agent interacting with the environment. At the beginning of each step, based on the current state S_t and the previous reward R_t, the agent performs an action A_t, transitioning to the state S_{t+1} while receiving reward R_{t+1}.

Now, using RL to learn dynamic robot control policies seem to be an obvious choice, but due to the painfully sequential nature of robots interacting with the real world, learning to solve complex tasks has been computationally infeasible in the past. However, with the arrival of GPU accelerated high-fidelity simulators - such as Flex, MuJoCo [TET12], and Bullet [CB] - we now have the possibility of training in a distributed setting without the risks and costs that working with physical robots usually entails. This has lead to some truly impressive feats, such as robots learning in-hand object reorientation using solely simulated training [Ope+18], and damaged robots learning to recover locomotive skills on the fly [CVM18].

Even though simulations have paved the way for distributed reinforcement learning in the field of robotics, we are still limited by how well we can model our reward functions. In [Zho+19] the concept of imitation learning is presented, where robots were shown to learn how to solve complex tasks based on expert examples. In other words, instead of designing a reward function, a human operator can simply 'show' the robot how the task should be performed and the robot can then infer the reward function based on how well it can mimic the expert. This work was made possible by the usage of *meta-learning* [FL17], which makes it possible to 'predict' an optimiser suitable for learning how to imitate the expert.

In the area of soft robotics, researchers are still somewhat limited by the supply of simulators capable of handling soft deformable objects, which might be why most soft robotics papers seem to focus on actuator modelling and quasi-static control tasks. In [Kri+19] a novel attempt at learning locomotive control for damaged soft robots where presented. Here, the authors used a distributed setting for exploring various policies simultaneously using evolutionary strategies - unfortunately, the prototype built to verify robot behaviour in the real world was not able to replicate the simulated behaviour, due to the limitations of the actuation devices used.

So far, the combination of reinforcement learning and simulators have produced impressive results, but for a lot of tasks, the learned control policies do not transfer directly from simulations to reality. Similar for most dynamic robotics tasks is that they rely on a vision-based system to acquire information about the state of the environment. Therefore, a challenging part of training RL agents using simulations is to replicate camera angles, light conditions, and camera settings, in the virtual environment. It is often not feasible to do this robustly, as the experimental set-up is prone to vary slightly between each session due to human error and out-of-control physical conditions.

To improve the quality of the learned control policy, there seem to be especially two approaches that researchers rely on: domain randomisation and domain adaptation. Domain randomisation is a technique which is used to deal with the irregularities of the real world by introducing a large amount of variability into the appearance of the simulated environment - i.e. change of colours, camera angles, light conditions, etc. and adding domain randomisation during training has been shown to increase the ability of the models to generalise to real world scenarios. Domain randomisation was even attributed as one of the key components in the success of the dexterous hand from [Ope+18]. In [JDJ17], control of a robotic gripper is shown to transfer directly from simulations, even when artificially high amount of noise is introduced in the real world. However, for some of the more difficult cases, the robotic gripper struggled to complete the grasping task.

To further bridge the gap between simulations and reality, real data seems to be needed during training. In [Kal+18] this was taken to the extreme, resulting in a grasping accuracy - on objects of varying material properties and appearances - of 96%. The authors captured footage of 608,000 grasps during off-policy training, using approximately 800 robot hours during a period of four months. In between the extreme and the methods that incorporate no real data, we find the concept of domain adaptation. The basic idea of domain adaptation is to adapt the parameters of simulations using real data to make the model of the physical environment more accurate. In [Che+19] a robotic arm was taught to complete various tasks through a mixture of simulated policy improvement and real world roll-outs. It was shown that a relatively small amount of roll-outs was necessary - 3 to be exact, for each of the tasks - reinforcing the validity of mixing real world data with simulated data.

In this thesis, time did not permit for us to implement any of these schemes, but based on the results of our experiments, the application of domain adaptation might have been a nice addition in order to decrease some of the discrepancies between our simulations and the real world behaviour of the soft robots. It should also be noted that what we refer to as transfer of a control policy in Experiment 7.3.3 is in fact just a replay of the control parameters found during minimisation within the simulation.

Chapter 8

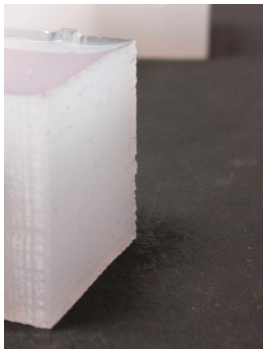
Summary

8.1 Discussion

Fabrication and Design

The robust manufacturing model initially seemed like a good idea, but in its current iteration it provides a too weak guarantee to be truly useful. We have observed that the model failed during the fabrication of the 'Bubblebot' robot, due to the vacuum created inside the mold, even though the robot design passed the robustness criteria. However, in terms of our cable driven robots, we were able to complete all fabrications without damaging any of the soft bodies, but it is difficult to determine if this was because the robots passed the robustness test or because the risk of failure for small robots is simply not present. As it is, our model is ill-suited for pressure-driven robot designs, but this was to be expected as single-piece molds appear to be unsuited for modelling the internal geometric structures of air-chambers.

For all of our robots we observed that the manufacturing process left undesirable artefacts in the soft robots. Observe in Figure 8.1 examples of these artefacts.



(a) Rippled surfaces due to the layered nature of 3D printed molds.

(b) Silicone 'traces' left by cable shelves.

(c) Air-bubbles that could not be removed completely during de-gassing.

Figure 8.1: Examples of the artefacts left by the current fabrication process.

These artefacts are introduced at different stages of the fabrication pipeline, and each artefact increases the gap between our simulations and reality. Especially the fabrication errors which

can be observed in Figure 8.1a and 8.1c introduce erroneous behaviour, as they alter frictional properties and material stiffness, respectively.

Another issue - which we have ignored in this thesis - is that the silicone bodies are unable to support their own weight in gravity, which causes the robots to deform unintentionally. To overcome this issue, all robots presented in this thesis have been hanging upside down, but in future iterations it might be beneficial to add an element of support to our robots. This support could be added as a 'skeleton' within the robot, or be an antagonistic actuator that can cancel out gravity when the robot is in its resting position.

The Learning Platform

The learning platform presented us with an opportunity to collect data of robots during actuation. We spent a substantial amount of time fabricating soft robots and rigs, but due to the work done in [Hol18], we were able to use the motor system to control cable-driven robots without any major complications. We ended up not using the marker-based segmentation of the learning platform to compare the simulated motion to the real robots being actuated, as complications with the Flex software limited our options substantially in terms of control policy optimisation. Instead, we only performed visual inspections of the quasi-static behaviour of the robots, for which only RGB images were needed. However, we still believe that if the control policies were to reach a level where the robots can interact dynamically with the environment, the marker-based segmentation will be well-suited for controlling the soft robots in real time. An interesting thing to note is that in our experience the colour based segmentation were more robust than the contrast based segmentation, but in [Hol18] it is actually commented that they observed the opposite behaviour. This emphasises the fact that there does not exists one perfect segmentation scheme, as the result depends too heavily on the variables of the system.

In [Ope+18], a construct similar to the Learning Cube was used to capture motion of the dexterous hand, which indicates that the general idea of transferring control policies to real robots using the Learning Cube is very much feasible. However, to achieve this we need to model the vision system inside of our simulations as well, including camera angles, light conditions, etc. Currently, our physical set-up allows us to extract the positions of markers placed on the robot, but more information might be required to complete difficult tasks. A natural next step for the vision system would be to extract the entire deformable object, as this would allow us to get a more accurate representation of the configuration of our soft robots.

Actuation Models

We formulated two force models for simulating our actuators of choice: cables and pneumatics. We modelled our cables as tendons, and observed that this might not be the most appropriate way to achieve realistic simulations of cables being pulled. Using a tendon representation meant that we left out a lot of the behaviour we would observe in real cables, such as friction and contact forces between the silicone and the cable itself. Another problem we noticed, was that our cables were limited in terms of discretisation options, which meant it would not always be possible to accurately measure the lengths of our cables. However, this was a limitation of the simulator and not the model.

On the other hand, the pneumatic model seemed to approximate the real world better, and in our initial experiments, the behaviour of the pressure forces seemed realistic, but in terms of the

visual results, we observed some discrepancies between the real world and our simulations. The pressure model depends on the assumption that the volume approximation scheme results in the correct ratios between current and initial volume during a simulation. To avoid having to make this assumption, we could have added 'ghost elements' corresponding to the elements of the mesh that would turn an air-chamber into a closed mesh. However, as we could not find a way to add these elements without doing it manually, we decided to use the approximation even though we knew it would not be entirely correct.

We implemented both models in the Flex simulator, as if it was a black box, by explicitly deriving positional updates for particles based on the forces computed by our models. This seem to work well for our cables, but for the pressure force simulations we observed that the forces were too large, leading to inverted triangles in our meshes. In our pressure model we took a 'shortcut' by only simulating forces as affecting the surface elements of the air-chambers. To make our simulations more realistic, we could have added an external pressure on the 'outside' of the robot corresponding to the initial pressure inside the robot. That we chose not to model the outside pressure might be one of the reasons that we observe the current behaviour in our pressure-force simulations, and it would be interesting to investigate this in future iterations of the pressure model.

The Reality Gap and the Exploration of Robot Work Spaces

At the beginning of this thesis, prototyping soft robot designs were a tedious process, which included manufacturing a lot of very similar robots. Thus, the design process was not particularly well suited for optimising a robot design, which is one of the reasons that the most exciting task our robots can perform is to move into a specified configuration. In Figure 8.2, the old design process can be found as a flow-chart.

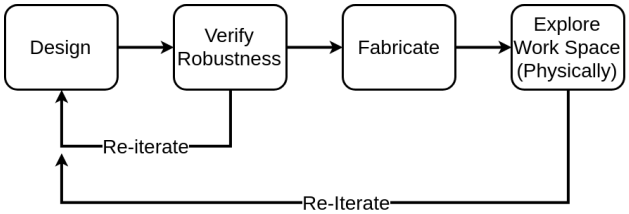


Figure 8.2: The old design process.

While the robustness test could weed out some robot designs that were bound to fail, we struggled a lot with the usability of our robots, as it was difficult to predict exactly how a slight design change would change the deformations the robot could achieve. The 'Hammerbot' was the first robot we created using the new design process, which is represented as a flow-chart in Figure 8.3.

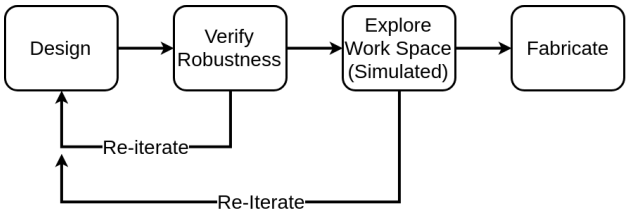


Figure 8.3: The new design process.

Even though the cable model lacks some sophistication, we were able to more or less explore how the robot worked prior to fabrication. We used the feedback from the simulations to alter the dimensions and cable placements of the robot to increase its reach. We believe that our cable model will increase the productivity of future students learning to design soft robots, as it increased our design capabilities significantly.

In terms of physical realism, our cable-driven simulations were lacking some accuracy, but this varied from robot to robot, and seemed to primarily be a meshing issue. The pressure force simulations on the other hand differed considerably from the physical behaviour of our pressure-driven robot, which means that we are still limited by the old design process for constructing our pneumatically actuated robots. The physical experiments also revealed that the physical components of the set-up introduces error sources that we have previously ignored, which introduced a small amount of hysteresis. It would be interesting to see if adding some degree of domain adaptation would make it possible to capture more of these variables in the simulated environment, but we would most likely need to alter our actuation models to include the addition of noise in that case.

Finally, we experienced that comparing simulated and actual robot motion was difficult using the current learning platform. This is primarily due to the fact that the vision system is not taken into account during simulation. Any comparisons are hence subject to the human errors introduced when we try to replicate the simulated environment inside the Learning Cube. We therefore believe that in future work, the simulation framework need to be updated to also simulate the vision-based part of the physical set-up.

Control Policy Optimisation

In Chapter 7, we showed that for well-behaved objective functions, we could identify the optimal control parameters needed to minimise the objective and the steepest descent algorithm was shown to be superior in terms of stability and reaching the minimum function value. The PID controller proved to converge quickly to a local minimiser, but due to a lack of fine-tuning of the gain parameters, the output signal performed worse than the control parameters found by the steepest descent approach. We only experimented on cable-driven robot designs, but it would be especially interesting to investigate if the same behaviour was present for the pneumatically actuated robots. The current pressure inside an object is given by,

$$P = \frac{nRT}{V} \tag{8.1}$$

and a physical pneumatic actuator - such as an air compressor - changes the concentration of air molecules - n - inside of a soft robot, which means that it acts as a multiplicative control scheme. This means that rapid changes to n should lead to large deformations in the robot, and therefore,

the PID controller might be better suited for learning a control scheme that would avoid large oscillations in the deformations of the soft body.

In our current implementation of pressure forces, we control the pressure using a multiplicative factor, $\beta^{(t)}$, as,

$$P^{(t)} = \beta^{(t)} \frac{P^{(0)} V^{(0)}}{V^{(t)}} - P^{(0)} \tag{8.2}$$

but the actuation device we are currently using - a syringe - changes the pressure by decreasing or increasing the volume of the system. We could have chosen to model this using an additive control parameter, $\gamma^{(t)}$, as,

$$P^{(t)} = \frac{P^{(0)}(V^{(0)} + \gamma^{(t)})}{V^{(t)}} - P^{(0)} \tag{8.3}$$

where a negative $\gamma^{(t)}$ corresponds to pumping air into the air-chamber, while a positive $\gamma^{(t)}$ corresponds to pulling the plunger and removing air. This additive control might have been better suited for the steepest descent algorithm, but we will have to leave a comparison for future work.

Originally, we would have liked to implement a more sophisticated optimisation strategy than steepest descent. Recent work in robotics have shown that evolution strategies (ES) are well suited for teaching robots to perform complex tasks [Che+19; Kri+19], and as detailed in [Lia+18], GPU-accelerated reinforcement learning should have been available in NVIDIA Flex. However, as the Flex simulator is still in its early stages - Alpha version 2.0.0 - it was not possible to use the promised reinforcement learning framework with our soft robots.

Due to the time limitations of this thesis, we only had time to create one set-up in which we could explore a policy transfer from the simulated environment to the real world. The task we choose was that of finding a configuration that resulted in the tip of the robot reaching a certain position - quite similarly to one of the case studies in [Hol18] and the case presented in [BKC17] - since the corresponding objective function is convex, making it easy to minimise. However, another reason we choose this objective function is that we have yet to find a 'real' task for which our soft robots are well-suited. Most tasks we have considered involves interacting with the environment in some capacity, but due to our current design and manufacturing abilities, we have been unable to create a robot that can perform any meaningful task well.

8.2 Conclusion

In this work, we have presented models for simulating cable and pressure forces in a position based dynamics simulator such as the Flex framework. To verify that our simulations resembled reality we fabricated physical copies of the digital models and compared the simulated motion to the real robots during actuation. To perform the comparison we applied the Learning Cube framework, but due to the nature of the data we needed to collect, we decided not to use the segmentation scheme developed for the Learning Cube. The cable model was shown to be robust and produced deformations approximating reality well. The pressure model, however, proved more volatile, and due to the crudeness of our physical actuation device, it was difficult to compare pressure simulations to real robot deformations.

We also presented two optimisation strategies - steepest descent and a PID controller - which we applied to an inverse kinematics problem using the cable model implemented in Flex. The steepest descent approach proved superior in terms of minimising the objective function, and the policy performed well when transferred to a real soft robot.

As expected, we observed several discrepancies between the simulations and reality, induced by factors such as motor drag, fabrication inaccuracies and flaws in the cable model. Due to these factors, future users of the Learning Cube and the design model from Chapter 2 should expect to observe some disparities between simulated and real behaviour, but we still believe that the tools developed as part of this thesis will increase the efficiency of future design iterations significantly.

8.3 Future Work

The Vision System

The vision system proved to have some major limitations when we wanted to compare simulated and real robot motion, since the set-up of cameras, robots, etc. are subject to human error. It would therefore be very beneficial to add some automation to the learning platform, such as cameras being able to replicate angles automatically and the robot rig being able to move into a specified position.

Currently, the cameras need to be calibrated by placing a number of reference points inside the box. It would be of great interest to add a 'calibration unit' to the cube, which when present in each frame could be used to create the mapping between cameras in real time. This would reduce the risk of experiments being invalidated by the cameras being knocked out of position during data collection, which unfortunately happened quite often.

Actuation Devices

The cable-driven system works rather well, but as the current pneumatic actuation system consists of a series of manually operated syringes, we need to improve this system significantly to be able to reliable transfer a policy from the simulated environment to the real world. However, so far we have not been able to find any pre-built systems that can be attached to the current learning framework, and it would require a substantial amount of time to develop the actuation system ourselves. It would be interesting to investigate the creation of a pneumatic system, driven by the stepper motors we already know how to use. These motors could be used to create a stepper motor 'piston pump', and it might be possible to create most of the parts needed using a 3D printer.

Actuation Models

As we observed in Chapter 6, our pneumatic simulations are a lot less reliable than the cable-driven simulations. In future work, the cause needs to be identified such that we can improve the model. We suspect that the error lies in the computation of air-chamber volumes, meshing of the digital models, or an unknown factor we have failed to take into account - or a combination of these issues. It would also be interesting to implement the pressure model in another simulator, to exclude the possibility of Flex being the cause of the poor simulations.

Policy Optimisation

Experimenting with more sophisticated objective functions and minimisers is a topic of great interest. We have only scratched the surface in terms of learning control policies, and in future work we would like to investigate if the robots can be 'taught' to interact with its environment. As

we did not achieve to get the distributed reinforcement learning part of Flex to work, this should be the first aim of any future iteration of this subject.

It would also be interesting to investigate the benefits of the PID controller more thoroughly when it is tuned more deliberately. Especially the usage of the PID controller in terms of controlling pneumatic behaviour is of interest.

Finally, the possibility of a fusion between the vision system and the control policy optimisation should be researched thoroughly, as any control policy learned will have to receive some kind of information about the environment - such as images or point clouds - if we want to solve dynamic tasks.

Design Optimisation

Instead of designing robots manually, we would like to make the process automated. That is, given an objective function to minimise, we would like to generate a robot which is able to minimise the objective. In [Che+14; Kri+19] a similar idea is investigated, and results were presented of robots learning to 'walk'. In these works robots were made of 'voxels' with specific properties, and the robot design space was very limited. We wanted to experiment with the generation of robots as CSG trees, as this would provide us with a vast design space which could be described using a relatively low amount of parameters. However, time did not permit us to investigate design optimisation thoroughly, but during the implementation of our control optimisations we identified some of the problems that need to be solved to make design optimisation feasible.

As a first step towards automated design generation, we need to be able to discard infeasible robot designs. In other words, we want to limit our search space to consist only of robots, we know can be fabricated. Originally, we wanted to use the robustness model to reject infeasible designs, but as we observed in Chapter 6, the guarantee given by the model was not as strong as we would have liked it to be. Thus, either the fidelity of the robustness model needs to be increased or another approach to verify that a robot can be fabricated, must be found before it really makes sense to begin optimising robot designs automatically.

Once a design has been deemed feasible and actuators have been placed, we then need to evaluate the performance of the robot. However, for each design we must first learn the optimal control parameters before we can determine the fitness of the design. Given the convergence rates for the simple task we minimised in Chapter 7, it seems computationally infeasible to evaluate designs this way, and we believe the possibility of optimising design and control simultaneously - as in [Che+14] - needs to be investigated in future work.

Bibliography

[AKY18] Alexander Alspach, Joohyung Kim, and Katsu Yamane. "Design and fabrication of a soft robotic hand and arm system". In: *2018 IEEE International Conference on Soft Robotics (RoboSoft)*. IEEE. 2018, pp. 369–375.

[Ard] Arduino. *Arduino — What is Arduino?* `https://www.arduino.cc/`. The Arduino web page, [Online; accessed 26-August-2019].

[AS] Universal Robots A/S. *Universal Robots — Address your labor needs, increase your productivity and become more competetive*. `https://www.universal-robots.com/`. The Universal Robots web page, [Online; accessed 21-August-2019].

[Åst02] Karl Johan Åström. "Control system design lecture notes for ME 155a, *Department of Mechanical and Environmental Engineering, University of California, Santa Barbara*". In: (2002), pp. 216–251.

[Ati12] Ahmad Atieh. "Design, Modeling, Fabrication and Testing of a Piezoresistive-Based Tactile Sensor for Minimally Invasive Surgery Applications". MA thesis. Concordia University, 2012.

[Atk89] Kendall E Atkinson. *An introduction to numerical analysis, 2nd Edition*. John Wiley & Sons, 1989. Chap. 5. Numerical Integration.

[Aut] Alias Systems Corporation Autodesk Inc. *Autodesk Maya — Overview*. `https://www.autodesk.com/products/maya/overview`. The Autodesk Maya web page, [Online; accessed 21-August-2019].

[BCC] James M Bern, Kai-Hung Chang, and Stelian Coros. "Interactive design of animated plushies". In: *ACM Trans. Graph. 36, 4, Article 80 (July 2017), 11 pages* (). DOI: `http://dx.doi.org/10.1145/3072959.3073700`.

[Ber+08] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 2008. Chap. 4 Linear Programming.

[Ber+19] James M. Bern, Pol Banzet, Roi Poranne, and Stelian Coros. "Trajectory Optimization for Cable-Driven Soft Robot Locomotion". In: *Robotics: Science and Systems 2019, Freiburg im Breisgau, June 22-26, 2019* (2019).

[BKC17] James M Bern, Grace Kumagai, and Stelian Coros. "Fabrication, modeling, and control of plush robots". In: (2017), pp. 3739–3746.

[Ble] Blender Foundation. *Blender — About — The Software*. `https://www.blender.org/about/`. The Blender web page, [Online; accessed 21-August-2019].

[BVa] 2011-2019 Ultimaker BV. *Ultimaker Cura Software — Software — , What makes Ultimaker Cura the most advanced 3D printer software?* `https://ultimaker.com/`. The Ultimaker Cura software web page, [Online; accessed 26-August-2019].

[BVb] 2011-2019 Ultimaker BV. *Ultimaker — Professional 3D printing made accessible.* `https://ultimaker.com/`. The Ultimaker web page, [Online; accessed 21-August-2019].

[CB] Erwin Coumans and Yunfei Bai. "PyBullet, a Python module for physics simulation for games, robotics and machine learning, 2017". In: *`http://pybullet.org`* ().

[CED17] Eulalie Coevoet, Adrien Escande, and Christian Duriez. "Optimization-based inverse model of soft robots with contact handling". In: *IEEE Robotics and Automation Letters* 2.3 (2017), pp. 1413–1419.

[Che+14] Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. "Unshackling evolution: evolving soft robots with multiple materials and a powerful generative encoding". In: *ACM SIGEVOlution* 7.1 (2014), pp. 11–23.

[Che+17] Desai Chen, David I. Levin, Wojciech Matusik, and Danny M. Kaufman. "Dynamics-Aware Numerical Coarsening for Fabrication Design". In: *ACM Trans. Graph., Article 84* 34.4 (2017). DOI: `10.1145/3072959.3073669`.

[Che+19] Yevgen Chebotar, Ankur Handa, Viktor Makoviychuk, Miles Macklin, Jan Issac, Nathan Ratliff, and Dieter Fox. "Closing the sim-to-real loop: Adapting simulation randomization with real world experience". In: *2019 International Conference on Robotics and Automation (ICRA)* (2019), pp. 8973–8979.

[Cor18] Intel Corporation. *Intel RealSense SDK 2.0.* `https://github.com/IntelRealSense/librealsense`. The Intel real sense software Github page, [Online; accessed 26-August-2019]. 2018.

[Cor19a] Intel Corporation. *Intel RealSense Depth Camera D415 — Details.* `https://click.intel.com/intelr-realsensetm-depth-camera-d415.html`. The Intel RealSense Cameras in the Intel Web Store, [Online; accessed 26-August-2019]. 2019.

[Cor19b] NVIDIA Corporation. *NVIDIA GameWorks — NVIDIA FleX 1.1.0 released.* `https://developer.nvidia.com/nvidia-flex-110-released`. We cannot currently link to NVIDIA Flex (Robotics Alpha 2.0.0), since it has not been released yet. Therefore, we refer to FleX 1.1.0 for a general idea about how FleX is distributed and how the source is exposed. 2019.

[Cra18] Keenan Crane. *AMS Short Course on Discrete Differential Geometry.* 2018. URL: `http://geometry.cs.cmu.edu/ddgshortcourse/AMSShortCourseDDG2018_Overview.pdf`.

[CVM18] Konstantinos Chatzilygeroudis, Vassilis Vassiliades, and Jean-Baptiste Mouret. "Reset-free trial-and-error learning for robot damage recovery". In: *Robotics and Autonomous Systems* 100 (2018), pp. 236–250.

[Dur13] Christian Duriez. "Control of elastic soft robots based on real-time finite element method". In: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 3982–3987.

[Ele19a] SparkFun Electronics. *EasyDriver - Stepper Motor Driver.* `https://www.sparkfun.com/products/12779`. 2019.

[Ele19b] SparkFun Electronics. *SparkFun RedBoard - Programmed with Arduino*. https://www.sparkfun.com/products/13975. 2019.

[Ele19c] SparkFun Electronics. *Stepper Motor with Cable*. https://www.sparkfun.com/products/9238. 2019.

[Eng12] Morten Pol Engell-Nørregård. *Interactive Modelling and Simulation of Human Motion*. 2012. Chap. III, Activation Splines, pp. 58–84.

[Erl+05] Kenny Erleben, Jon Sporring, Knud Henriksen, and Henrik Dohlmann. *Physics-Based Animation*. 2005. Chap. 8.4.4, pp. 281–284.

[FL17] Chelsea Finn and Sergey Levine. "Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm". In: *arXiv preprint arXiv:1710.11622* (2017). Published as a conference paper at ICLR 2018.

[Gal+16] Kevin C Galloway, Kaitlyn P Becker, Brennan Phillips, Jordan Kirby, Stephen Licht, Dan Tchernov, Robert J Wood, and David F Gruber. "Soft robotic grippers for biological sampling on deep reefs". In: *Soft robotics* 3.1 (2016), pp. 23–33.

[Gas+19] Renato Gasoto, Miles Macklin, Xuan Liu, Yinan Sun, Kenny Erleben, Cagdas D. Onal, and Jie Fu. "A Validated Physical Model For Real-Time Simulation of Soft Robotic Snakes". In: *CoRR* abs/1904.02833 (2019). Paper accepted on ICRA 2019. arXiv: 1904.02833. URL: http://arxiv.org/abs/1904.02833.

[Gli+18] Paul Glick, Srinivasan A Suresh, Donald Ruffatto, Mark Cutkosky, Michael T Tolley, and Aaron Parness. "A Soft Robotic Gripper With Gecko-Inspired Adhesive". In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 903–910.

[Hao+17] Yufei Hao, Tianmiao Wang, Ziyu Ren, Zheyuan Gong, Hui Wang, Xingbang Yang, Shaoya Guan, and Li Wen. "Modeling and experiments of a soft robotic gripper in amphibious environments". In: *International Journal of Advanced Robotic Systems* 14.3 (2017), p. 1729881417707148.

[HE18] Fredrik Holsten and Kenny Erleben. *pySoRo*. https://github.com/erleben/pySoRo. The pySoRo software package github repository, [Online; accessed 26-August-2019]. 2018.

[HHZ19] Zhewei Huang, Wen Heng, and Shuchang Zhou. "Learning to Paint with Model-based Deep Reinforcement Learning". In: *CoRR* abs/1903.04411 (2019). arXiv: 1903.04411. URL: http://arxiv.org/abs/1903.04411.

[HL12] Jonathan Hiller and Hod Lipson. "Automatic design and manufacture of soft robots". In: *IEEE Transactions on Robotics* 28.2 (2012), pp. 457–466.

[Hol+18] Fredrik Holsten, Sune Darkner, Morten Pol Engell-Nørregård, and Kenny Erleben. "Local models for data driven inverse kinematics of soft robots". In: *Proceedings of the Conference on Computer Animation* (2018), pp. 7–8.

[Hol18] Fredrik Holsten. "Soft Robotics AI". MA thesis. University of Copenhagen, 2018.

[Ili+11] Filip Ilievski, Aaron D Mazzeo, Robert F Shepherd, Xin Chen, and George M Whitesides. "Soft robotics for chemists". In: *Angewandte Chemie* 123.8 (2011), pp. 1930–1935.

[Inc18] Mann Release Technologies Inc. *Ease Release^{TM} 200*. https://www.mann-release.com/products/ease-release/200/. The Ease Releae product web page, [Online; accessed 26-August-2019]. 2018.

[Ind] MakerBot Industries. *Makerbot — Products — MAKERBOT 3D PRINTERS*. https://www.makerbot.com/3d-printers/. The MakerBot web page, [Online; accessed 26-August-2019].

[Jac+18] Alec Jacobson, Daniele Panozzo, Christian Schüller, Olga Diamanti, Qingnan Zhou, Sebastian Koch, Jeremie Dumas, Amir Vaxman, Nico Pietroni, Stefan Brugger, Kenshi Takayama, Wenzel Jakob, Nikolas De Giorgis, Luigi Rocca, Leonardo Sacht, Kevin Walliman, and Olga Sorkine-Hornung. *libigl*. libigl.github.io. The libigl software package. 2018.

[Jäh05] Bernd Jähne. *Digital Image Processing*. Springer, 2005. Chap. 18 Morphology, pp. 501–514.

[JDJ17] Stephen James, Andrew J. Davison, and Edward Johns. "Transferring End-to-End Visuomotor Control from Simulation to Real World for a Multi-Stage Task". In: *CoRR* abs/1707.02267 (2017). 1st Conference on Robot Learning (CoRL 2017), Mountain View, United States. arXiv: 1707.02267. URL: http://arxiv.org/abs/1707.02267.

[Kal+18] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. "QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation". In: *CoRR* abs/1806.10293 (2018). CoRL 2018 camera ready. 23 pages, 14 figures. arXiv: 1806.10293. URL: http://arxiv.org/abs/1806.10293.

[Kri+19] Sam Kriegman, Stephanie Walker, Dylan Shah, Michael Levin, Rebecca Kramer-Bottiglio, and Josh Bongard. "Automated shapeshifting for function recovery in damaged robots". In: *Robotics: Science and Systems 2019, Freiburg im Breisgau, June 22-26, 2019* (May 2019).

[KW10] Marius Kintel and Clifford Wolf. *OpenSCAD — The Programmers Solid 3D CAD Modeller*. www.openscad.org. OpenSCAD Web Page, [Online; accessed 21-August-2019]. 2010.

[Lee+17] Chiwon Lee, Myungjoon Kim, Yoon Jae Kim, Nhayoung Hong, Seungwan Ryu, H Jin Kim, and Sungwan Kim. "Soft robot review". In: *International Journal of Control, Automation and Systems* 15.1 (2017), pp. 3–15.

[Lev+16] Sergey Levine, Peter Pastor, Alex Krizhevsky, and Deirdre Quillen. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection". In: *International Symposium on Experimental Robotics*. Springer. 2016, pp. 173–184.

[Lia+18] Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, and Dieter Fox. "GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning". In: *CoRR* abs/1810.05762 (2018). Accepted and to appear at the Conference on Robot Learning (CoRL) 2018. arXiv: 1810.05762. URL: http://arxiv.org/abs/1810.05762.

[LLT11] Huai-Ti Lin, Gary G Leisk, and Barry Trimmer. "GoQBot: a caterpillar-inspired soft-bodied rolling robot". In: *Bioinspiration & biomimetics* 6.2 (2011), p. 026007.

[Ltd] Aniwaa Pte. Ltd. *StrataSys — Products — J735 AND J750*. https://www.stratasys.com/3d-printers/j735-j750. The Stratasys Products Page, [Online; accessed 26-August-2019].

[Ltd19] Aniwaa Pte. Ltd. *StrataSys — Products — J750 STRATASYS*. https://www.aniwaa.com/product/3d-printers/stratasys-j750/. The Stratasys J750 Page, [Online; accessed 26-August-2019]. 2019.

[Mac+19] Miles Macklin, Kenny Erleben, Matthias Müller, Nuttapong Chentavez, Stefan Jeschke, and Viktor Makovivychuk. "Non-Smooth Newton Methods for Deformable Multi-Body Dynamics". In: *ACM Trans. Graph 38, 5, Article 20 (June 2019), 20 pages* (2019).

[Mat97] Maja J Matarić. "Reinforcement learning in the multi-robot domain". In: *Robot colonies*. Springer, 1997, pp. 73–83.

[Meg+17] Vittorio Megaro, Espen Knoop, Andrew Spielberg, David IW Levin, Wojciech Matusik, Markus Gross, Bernhard Thomaszewski, and Moritz Bächer. "Designing cable-driven actuation networks for kinematic chains and trees". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM. 2017, p. 15.

[Mis+12] Marek Krzysztof Misztal, Kenny Erleben, Adam Bargteil, Jens Fursund, B Christensen, Jakob Andreas Bærentzen, and Robert Bridson. "Multiphase flow of immiscible fluids on unstructured moving meshes". In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Eurographics Association. 2012, pp. 97–106.

[MMC16] Miles Macklin, Matthias Müller, and Nuttapong Chentanez. "XPBD: position-based simulation of compliant constrained dynamics". In: *Proceedings of the 9th International Conference on Motion in Games*. ACM. 2016, pp. 49–54.

[MO03] Maciej Matyka and Mark Ollila. "Pressure model of soft body simulation". In: *The Annual SIGRAD Conference. Special Theme-Real-Time Simulations. Conference Proceedings from SIGRAD2003*. 010. Linköping University Electronic Press. 2003, pp. 29–33.

[Mül+07] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. "Position based dynamics". In: *Journal of Visual Communication and Image Representation* 18.2 (2007), pp. 109–118.

[NW06a] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[NW06b] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006. Chap. 3. Line Search Methods.

[NW06c] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006. Chap. 8. Calculating Derivatives.

[Ope+18] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafał Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. "Learning Dexterous In-Hand Manipulation". In: *CoRR* (2018). URL: http://arxiv.org/abs/1808.00177.

[Pac+] Jakub Pachocki, David Farhi, Szymon Sidor, Greg Brockman, et al. *OpenAI Five — Overveiw*. `www.openai.com/five`. The OpenAI Five web page, [Online; accessed 21-August-2019].

[Par+19] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. "Semantic image synthesis with spatially-adaptive normalization". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2019), pp. 2337–2346.

[PC99] Michael Peshkin and J Edward Colgate. "Cobots". In: *Industrial Robot: An International Journal* 26.5 (1999), pp. 335–341.

[PRI18] WANHAO 3D PRINTER. *WANHAO DUPLICTOR I3 V2.1*. `http://www.wanhao3dprinter.com/Unboxin/ShowArticle.asp?ArticleID=70`. 2018.

[RCD17] Alejandro Rodríguez, Eulalie Coevoet, and Christian Duriez. "Real-time simulation of hydraulic components for interactive control of soft robots". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2017, pp. 4953–4958.

[Rey+13] Thierry Rey, Grégory Chagnon, Jean-Benoit Le Cam, and Denis Favier. "Effects of temperature on the mechanical behavior of filled and unfilled silicone rubbers". In: *8th European Conference on Constitutive Models for Rubbers, Jun 2013, San Sebastian, Spain* (2013), pp. 511–514.

[Rie+09] Martin Ried-Miller, Thomas Gabel, Roland Hafner, and Sascha Lange. "Reinforcement learning for robot soccer". In: *Autonomous Robots* 27.1 (2009), pp. 55–73.

[RT15] Daniela Rus and Michael T Tolley. "Design, fabrication and control of soft robots". In: *Nature* Volume 521 (May 2015).

[SB18] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[She+11] Robert F Shepherd, Filip Ilievski, Wonjae Choi, Stephen A Morin, Adam A Stokes, Aaron D Mazzeo, Xin Chen, Michael Wang, and George M Whitesides. "Multigait soft robot". In: *Proceedings of the national academy of sciences* 108.51 (2011), pp. 20400–20403.

[She02] Jonathan Richard Shewchuk. "What is a good linear finite element? interpolation, conditioning, anisotropy, and quality measures (preprint)". In: *University of California at Berkeley* 73 (2002), p. 137.

[Si] Hang Si. *TetGen — A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator*. `www.wias-berlin.de/software/tetgen/`. The TetGen web page, [Online; accessed 21-August-2019].

[Sin+19] Avi Singh, Larry Yang, Chelsea Finn, and Sergey Levine. "End-To-End Robotic Reinforcement Learning without Reward Engineering". In: *Proceedings of Robotics: Science and Systems*. Freiburg im Breisgau, Germany, June 2019. DOI: `10.15607/RSS.2019.XV.073`.

[SIT91] Koichi Suzumori, Shoichi Iikura, and Hirohisa Tanaka. "Development of flexible microactuator and its applications to robotic mechanisms". In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. IEEE. 1991, pp. 1622–1627.

[Smo] Inc. Smooth-On. *Smooth-On — Products — Ecoflex^{TM} 00-50*. `https://www.smooth-on.com/products/ecoflex-00-50/`. The Ecoflex 00-50 silicone from the Smooth-On Products page, [Online; accessed 21-August-2019].

[TET12] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 5026–5033.

[Too18] Soft Robotics Toolkit. *SDM fingers*. `https://softroboticstoolkit.com/book/sdm-fingers`. 2018.

[Tre83] Jay S Treiman. "Characterization of Clarke's tangent and normal cones in finite and infinite dimensions". In: *Nonlinear Analysis: Theory, Methods & Applications* 7.7 (1983), pp. 771–783.

[ZC01] Cha Zhang and Tsuhan Chen. "Efficient feature extraction for 2D/3D objects in mesh representation". In: *Proceedings 2001 International Conference on Image Processing (Cat. No. 01CH37205)*. Vol. 3. IEEE. 2001, pp. 935–938.

[Zha+19] Zhongkai Zhang, Jeremie Dequidt, Junghwan Back, Hongbin Liu, and Christian Duriez. "Motion Control of Cable-Driven Continuum Catheter Robot Through Contacts". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 1852–1859.

[Zho+19] Allan Zhou, Eric Jang, Daniel Kappler, Alexander Herzog, Mohi Khansari, Paul Wohlhart, Yunfei Bai, Mrinal Kalakrishnan, Sergey Levine, and Chelsea Finn. "Watch, Try, Learn: Meta-Learning from Demonstrations and Reward". In: *CoRR* abs/1906.03352 (2019). arXiv: `1906.03352`. URL: `http://arxiv.org/abs/1906.03352`.

Appendices

Appendix A

Soft Robots

A.1 Robot 0

A soft robot inspired by the silicone finger from *Soft Robotics Toolkit* [Too18] developed for the master thesis [Hol18]. The finger is made of a two component silicone which hardens when mixed together. A mould was made out of cardboard, in which tubing was placed that cables used for actuation could go through. A metal plate with screw holes was also added to the mould, such that the robot could be attached to Rig B.1. To keep the wire in place, it was super glued to the inside of last joint of the finger.

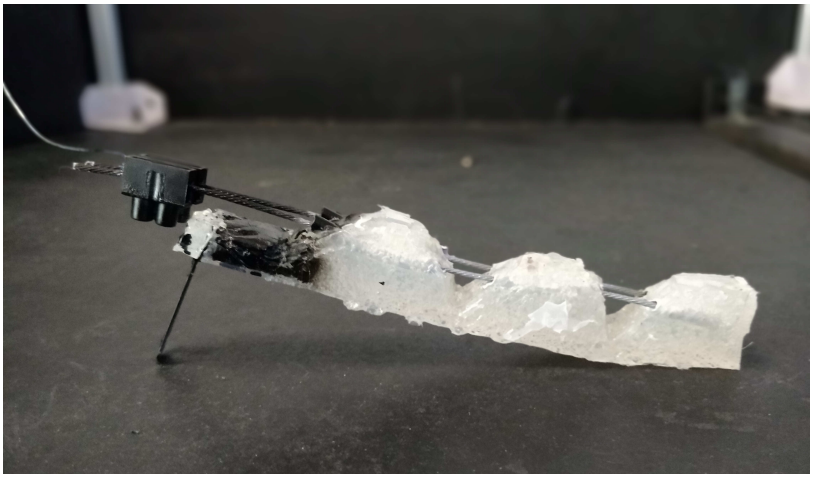


Figure A.1: The soft robot built for [Hol18].

A.2 Robot 1

For this robot we were inspired by the soft robot presented in [CED17]. The robot gripper was made using the same silicone as Robot A.1 and using the same proportion between the two components of the silicone. The robot can be actuated by three motors and contains cable tunnels going through the upper part of the robot and through both sides, through all wedges except the thinnest one at the end. For cable sheathing we used black plastic straws, and for keeping the wire in place we used small washers. The robot was equipped by a single red marker at the tip.

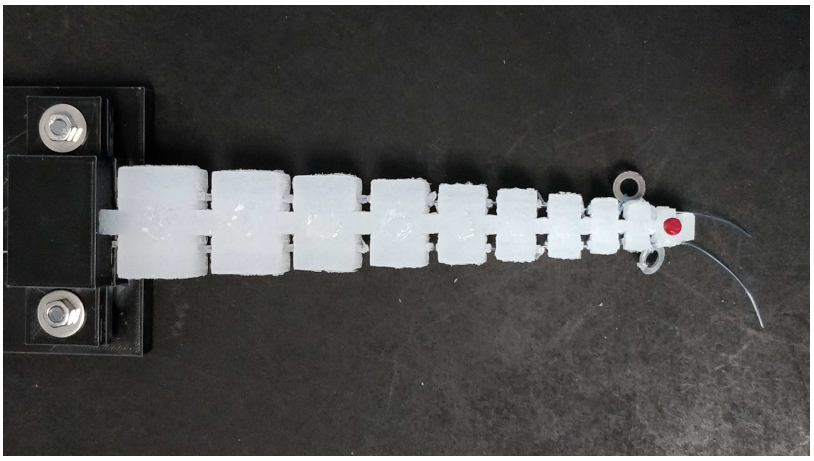


Figure A.2: The soft robot used in Chapter 3.

The robot was originally attached to Rig B.1 by inserting a screw directly into the silicone in the larger end, but this method is not very sustainable if the robot is to be used for an extended period of time. Therefore, Rig B.2 was built to accommodate the robot by locking the large end of the robot into place, as can be seen in Figure B.3 in Appendix B.2. This rig - along with the robot - was used for the motion capture experiments in Section 3.

A.3 The Hammerbot

The 'Hammerbot' robot from Chapters 6 and 7. The robot was attached to rig B.3.

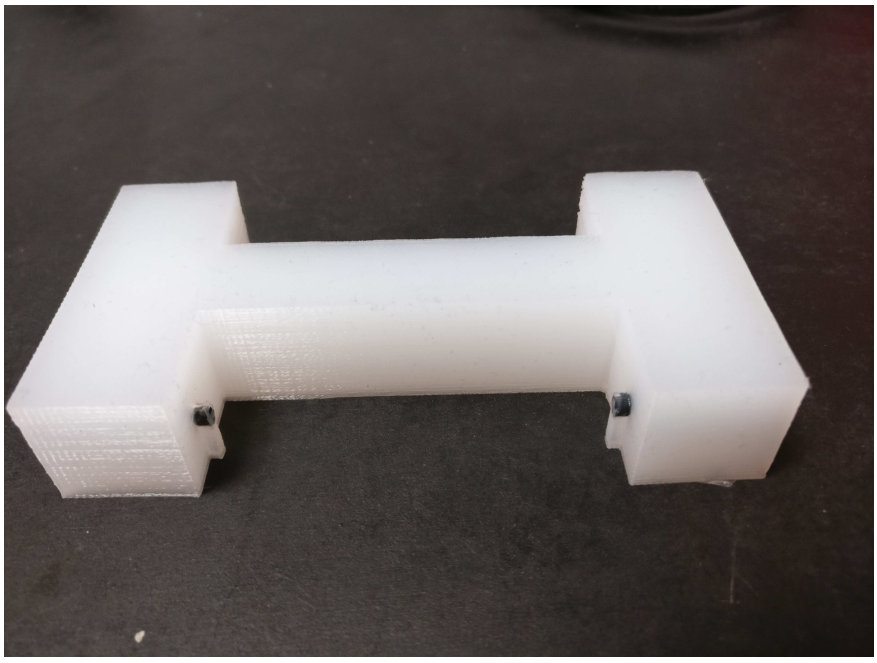


Figure A.3: One of the soft robots used in Chapters 6 and 7.

A.4 The Spinebot

The ‘Spinebot’ robot from Chapters 6 and 7. The robot was attached to rig B.3.

Figure A.4: One of the soft robots used in Chapters 6 and 7.

A.5 The Bubblebot

The ‘Bubblebot’ robot from Chapters 6 and 7.

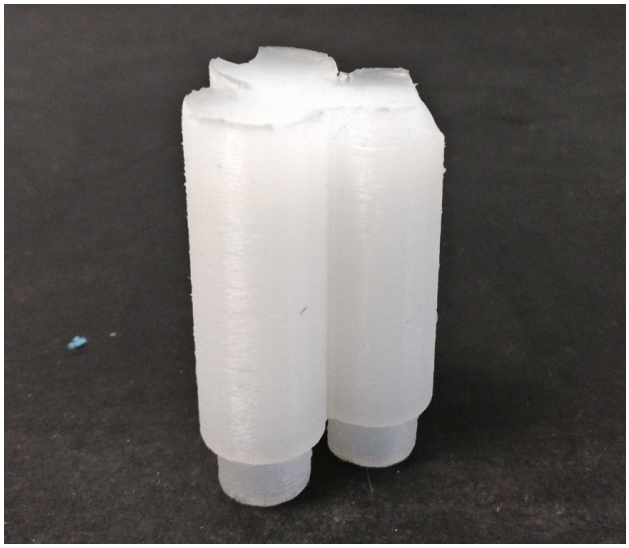


Figure A.5: One of the soft robots used in Chapters 6 and 7.

Unlike the other robots, this robot was attached to a ‘motor house’ that connected the robot to the actuators. The motor house can be seen in rig B.4

Appendix B

Rigs

B.1 Rig 1

The original rig built for use with Robot A.1 as part of [Hol18]. The rig can be equipped by a single motor and can be translated by another motor by using the rails, which can be found in Figure B.2

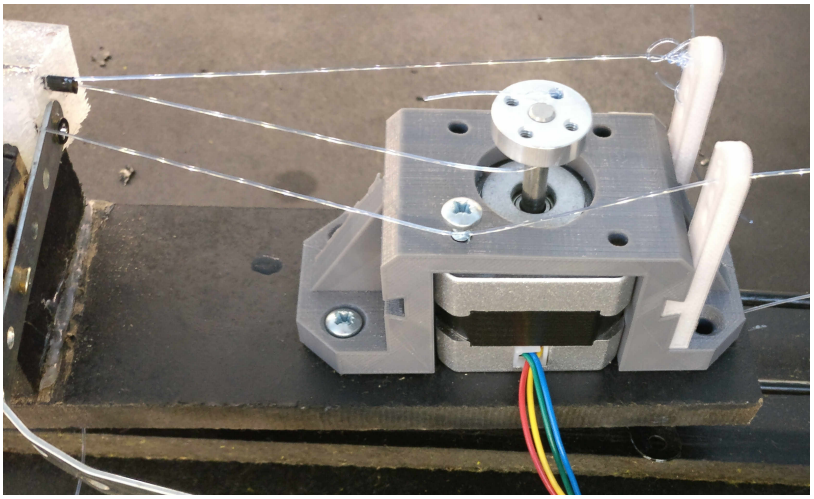


Figure B.1: The rig created for Robot A.1.

Figure B.2: The rails which Rig B.1 can use to achieve translational motion.

B.2 Rig 2

The rig created specifically to accommodate Robot A.2.

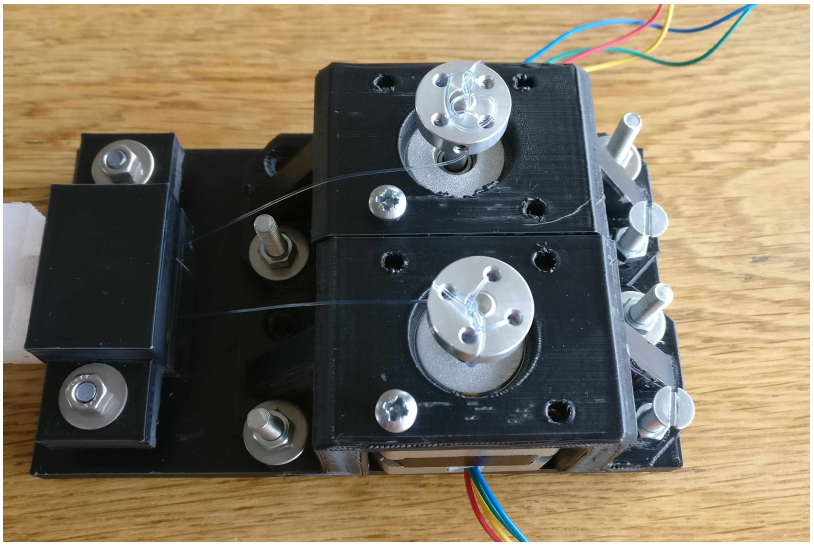


Figure B.3: The rig created for Robot A.2.

B.3 Rig 3

The rig created to accommodate robots A.3 and A.4.

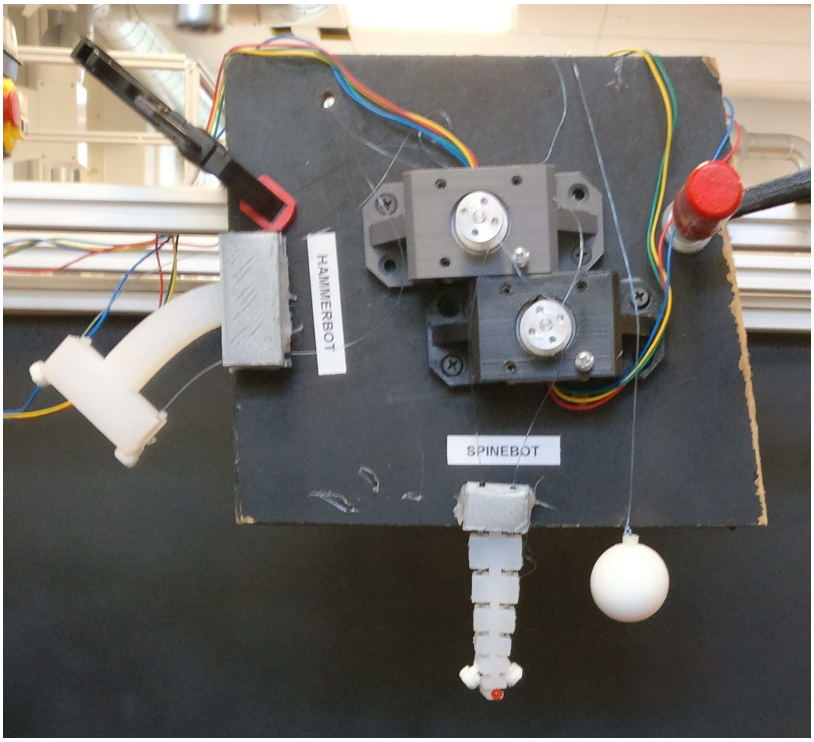


Figure B.4: The rig created for robots A.3 and A.4.

B.4 Rig 4

The ‘motor house’ of the ‘Bubblebot’ robot.

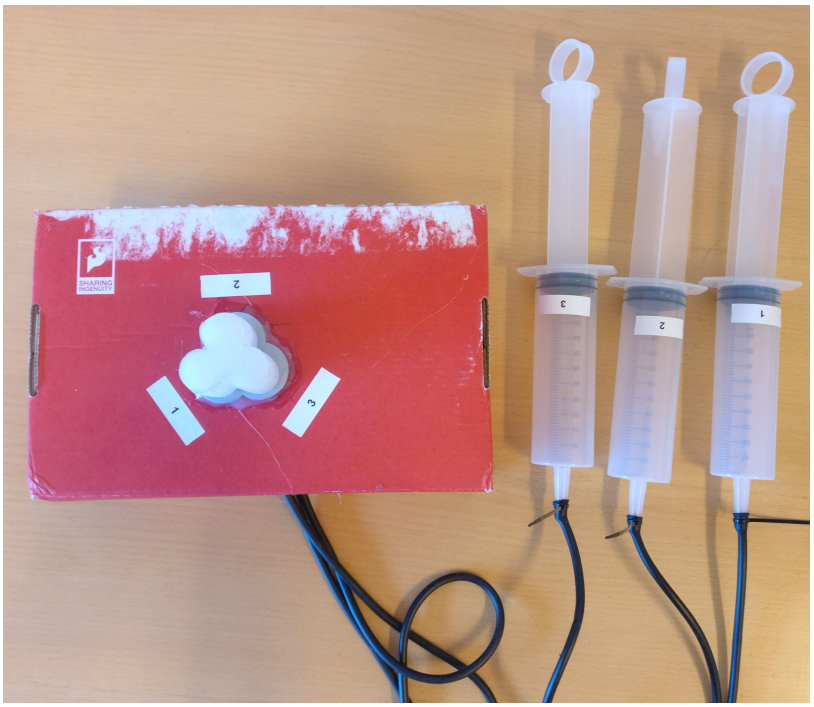


Figure B.5: The rig created for robot A.5.

Appendix C

Learning Goals

1. Briefly describe how to use the learning cube to collect data observations of the motion of a soft robot.

2. Formulate a model for cable actuators (computing cable actuator forces) and describe a design of how to implement the model into the NVIDIA FleX simulator.

3. Formulate a model for pneumatic control (computing pressure fields) and describe a design of how to implement the model into the NVIDIA FleX simulator.

4. Present results of how one can use the design model developed in pre-project to create (instantiate) a simulation model in FleX and interactively explore the configuration and workspace of the soft robot.

5. Present results of using the design model from pre-project on manufacturing a real soft robot, reflect on the usefulness of the model obtained from pre-project. If time permits possible compare the real robot motion with the simulated ones from (4) to reflect on physical realism of the simulation model in (4). The approach from (1) will be intended for 'getting motion' of the real robot.

6. Design a model for how to apply optimisation or learning to control a simulated soft robot model from (4) to solve a simple identified task and/or optimise. If time permits then compare the simulated motion control to the real robot from (5). Comparison should be used to evaluate if the learned control signals was successful on the real robot. The approach from (1) will be intended for comparison purposes.

7. If time permits, design a model for how to apply optimisation or learning to optimise the design model of a soft robot using the simulation model from (4) to solve a simple identified task. If time permits then manufacture the optimised design and compare the real the optimised robot with the non-optimised robot from (5). Comparison should be used to evaluate if the optimisation was successful. The approach from (1) will be intended for comparison purposes.