ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

RSL Robotic Systems Lab
Prof. Dr. Marco Hutter

Bachelor Thesis

# Development of a Robotic Gripper for Microgravity Applications

**Autumn Term 2024**

**Supervised by:**
Joseph Church
Robert Baines

**Authors:**
Fiona Martin
Simon Ramchandani

# Declaration of Originality

I hereby declare that the written work I have submitted entitled

**Development of a Robotic Gripper for Microgravity Applications**

is original work which I alone have authored and which is written in my own words.[1]

**Author(s)**

| | |
|---|---|
| Fiona | Martin |
| Simon | Ramchandani |

**Student supervisor(s)**

| | |
|---|---|
| Joseph | Church |
| Robert | Baines |

**Supervising lecturer**

| | |
|---|---|
| Marco | Hutter |

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/plagiarism-citationetiquette.pdf`). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Neftenbach 24.06.24
Place and date


Signature

[1] Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

# Intellectual Property Agreement

The student acted under the supervision of Prof. Hutter and contributed to research of his group. Research results of students outside the scope of an employment contract with ETH Zurich belong to the students themselves. The results of the student within the present thesis shall be exploited by ETH Zurich, possibly together with results of other contributors in the same field. To facilitate and to enable a common exploitation of all combined research results, the student hereby assigns his rights to the research results to ETH Zurich. In exchange, the student shall be treated like an employee of ETH Zurich with respect to any income generated due to the research results.
This agreement regulates the rights to the created research results.

### 1. Intellectual Property Rights

1. The student assigns his/her rights to the research results, including inventions and works protected by copyright, but not including his moral rights ("Urheberpersönlichkeitsrechte"), to ETH Zurich. Herewith, he cedes, in particular, all rights for commercial exploitations of research results to ETH Zurich. He is doing this voluntarily and with full awareness, in order to facilitate the commercial exploitation of the created Research Results. The student's moral rights ("Urheberpersönlichkeitsrechte") shall not be affected by this assignment.
2. In exchange, the student will be compensated by ETH Zurich in the case of income through the commercial exploitation of research results. Compensation will be made as if the student was an employee of ETH Zurich and according to the guidelines "Richtlinien für die wirtschaftliche Verwertung von Forschungsergebnissen der ETH Zürich".
3. The student agrees to keep all research results confidential. This obligation to confidentiality shall persist until he or she is informed by ETH Zurich that the intellectual property rights to the research results have been protected through patent applications or other adequate measures or that no protection is sought, but not longer than 12 months after the collaborator has signed this agreement.
4. If a patent application is filed for an invention based on the research results, the student will duly provide all necessary signatures. He/she also agrees to be available whenever his aid is necessary in the course of the patent application process, e.g. to respond to questions of patent examiners or the like.

### 2. Settlement of Disagreements

Should disagreements arise out between the parties, the parties will make an effort to settle them between them in good faith. In case of failure of these agreements, Swiss Law shall be applied and the Courts of Zurich shall have exclusive jurisdiction.

Neftenbach, 24.06.24
Place and date


Signature

# Contents

# Acknowledgements

We would like to express our gratitude and sincere thanks towards our supervisors Joseph Church and Robert Baines. Their valuable expertise has helped us immensely along the path of this thesis. We would also like to thank Prof. Dr. Marco Hutter and the whole Robotic Systems Lab for providing us with the support needed to accomplish this project. Special thanks go to Rene Zurbruegg for his assistance with the Franka arm and for providing the initial code framework. Additionally, we are grateful to the Spacehopper team for providing us with valuable information about the SpaceHopper.

Lastly, we want to thank our friends and family who have supported us throughout this project.

# Abstract

The exploration of asteroids and other extraterrestrial objects is increasingly significant in space research. Deploying robots in these micro-gravitational environments to gather surface and composition data provides valuable insights into the history of bodies in our solar system. However, these insights are often found in inaccessible environments such as crater walls or cliffs, making it impossible to reach these spots of interest with wheeled or legged rovers which lack climbing features. The goal of this project is to contribute to ongoing research by developing a robotic climbing gripper designed for microgravity conditions, which can be integrated into robotic platforms. Different gripping prototypes were developed using the CAD software SIEMENS NX. The concepts were 3D printed and tested in the course of the project. Three gripper legs were implemented per gripper, each containing a gripping foot which houses spines for attaching to the surface, and a linkage system connecting the gripping foot to the motor housing. Instead of microspines, nails of $1mm$ diameter were used. Design parameters were tested on six types of rock, representing various topologies the gripper might encounter on an exploratory mission in space. Using the robotic *Franka* arm, which was provided by the Robotic Systems Lab of ETH Zurich, an optimal spine angle of $50°$ with respect to the surface was found, producing a vertical pulling force of $3.5N$ when utilising $5N$ tangential (to the ground) force. The final design was tested by determining the effective pulling force of the attached gripper. This was done by manually measuring how much force could be applied to the gripper before detachment from the rock would occur. A maximum pulling force of $14.42N$ was found, which enables the gripper to successfully attach to surfaces in environments with a gravitational acceleration lower than $5m(s^{-2})$. This assumes that the gripper is implemented into the robot SpaceHopper and that all gripping legs are attached to the target surface while the robot is hanging inverted on a ceiling.

# Symbols

## Symbols

| | |
|---|---|
| # | Amount of / Number of |
| $F$ | Force [N] |
| $F_s$ | Spring Force [N] |
| $F_x$ | Force in x-direction [N] |
| $F_z$ | Force in z-direction [N] |
| $g$ | Earth's gravitational acceleration 9.91 $[m(s^{-2})]$ |
| $\alpha$ | Spine Incident Angle [°] |
| $r_s$ | Spine Tip Radius [m] |
| $k$ | Spring Constant $[N(mm)^{-1}]$ |
| $n$ | Number of Spines |
| $m$ | Mass [kg] |

## Acronyms and Abbreviations

| | |
|---|---|
| ETH | Eidgenössische Technische Hochschule |
| JPL | Jet Propulsion Laboratory |
| RSL | Robotic Systems Lab |
| DOF | Degree[s] Of Freedom |
| wrt | With Respect To |

# Chapter 1

# Introduction

## 1.1 Motivation

The exploration of asteroids and other extraterrestrial objects has become an increasingly significant focus within the field of space research. Deploying robots in these micro-gravitational environments to gather information about the surface and composition of celestial bodies can provide valuable insights into the formation of our solar system [1]. Asteroids also pose a potentially catastrophic threat to our civilization. The study of John Brophy et al. [2] explored the concept of an Asteroid Redirect Mission (ARM), which could address this problem. Future missions aimed at landing on and taking measurements of these rocks will be crucial for the success of such endeavors.
Rovers and robots attempting to land in these micro-gravitational environments face significant challenges in locomotion, anchoring, and sample collection. The low gravity and irregular surface features make traditional locomotion methods, such as wheels or tracks, ineffective, unreliable, or even impossible. To tackle these challenges and successfully navigate this difficult terrain new technologies must be developed and implemented.
The motivation behind this research is to develop a modular robotic gripper that can be mounted as feet on various robots, such as the SpaceHopper [3]. This gripper will enable these robots to land on and anchor to consolidated rocks on asteroids. Providing the robots with reliable means of attachment to the surface through the gripper, they would be able to perform scientific measurements, collect samples, and conduct other critical tasks more effectively and safely.

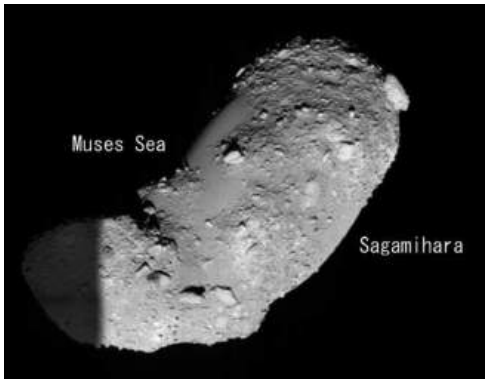


Figure 1.1: Rubble-pile asteroid Itokowa. *Credit: A. Fujiwara et al., Science, 2006*

An example of the types of surfaces these robots may encounter is depicted in Figure 1.1 of the Itokawa asteroid [4]. Although rubble-pile asteroids are thought to be mostly a loose aggregation of rocks, one can see large boulders on the surface of the asteroid [1], on which landing and anchoring could be possible.
Figure 1.2 shows examples of features on extra-terrestrial bodies that can only hardly be reached with traditional locomotion. The left picture shows a martian crater on a steep hill. The middle picture shows an icy cliff on mars revealing some of the planets history in the layers of ice. The right picture depicts the entrance of a lunar lava tube, a feature on our moon which has not been explored much due to its inaccessibility.

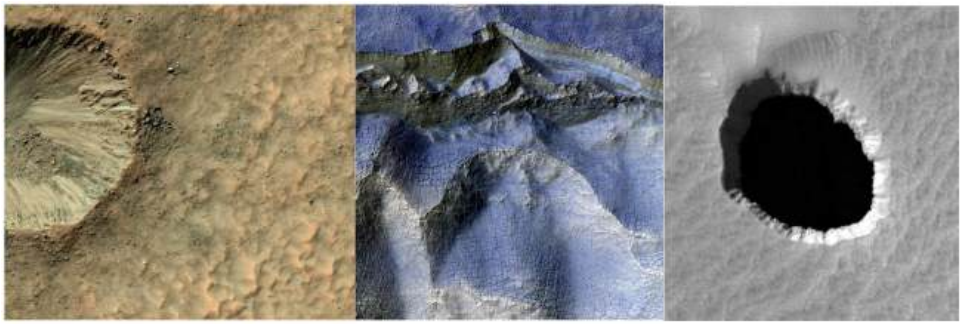

Figure 1.2: Left: Martian crater on a steep hill. Middle: Cliffs in ancient ice on mars. Right: Entrance of a lunar lava tube. *Credit: NASA/JPL-Caltech/University of Arizona 2017, E. Ravanis, Geobites 2020*

For the reasons mentioned above, the goal of this thesis is to develop a robotic gripper which can be used for the exploration of challenging terrains.

## 1.2 State of the Art

### 1.2.1 Gripping Mechanisms

A promising field of application for robotic gripping mechanisms is the exploration of bodies in space, for example asteroids or exoplanets. Developing climbing and grasping robots gives the possibility to explore new terrains that are not accessible by conventional locomotion such as rolling and walking. In this section, different existing gripping mechanisms will be reviewed. In doing so, an understanding of relevant design features and restrictions shall be established.

#### Example Macroscale Gripping - JPL Nautilus Gripper

The JPL Nautilus gripper was developed for underwater operations, namely, sampling rocks from the sea floor. To fulfill this purpose, an active, underactuated gripper was designed, featuring 16 axisymmetrical arms. The end of each arm is composed of a hook which can attach to soft rocks [5].

Figure 1.3: JPL Nautilus Gripper grasping a ball during testing. *Credit: Backus et al. Design and testing of the JPL-Nautilus Gripper for deep-ocean geological sampling. J Field Robotics 2020*

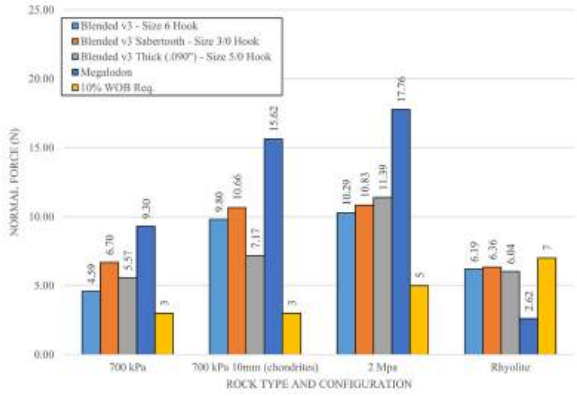


Figure 1.4: Normal gripping force for various hooks gripping into different stone types. *Credit: Backus et al. Design and testing of the JPL-Nautilus Gripper for deep-ocean geological sampling. J Field Robotics 2020*

One apparent design feature of the gripper is its hook size and high number of arms. Those parameters were determined by testing how much normal force different hook sizes would produce on different rock surfaces. This gives a first insight into the complexity of spine design, which will be further elaborated in section 2.2.1. The gripper is actuated by a differential pulley mechanism, distributing the loads equally over all 16 arms [5].

#### Example Microscale Gripping - Stanford Microspine Gripper

When looking at microspine technology, the Stanford microspine gripper [6] is an excellent example for spine integration. The gripper consists of multiple microspine tiles, which each contain 60 spines. The spine tip radius $r_s$ is small enough to fit into microscale asperities in the rock surface. By applying a force tangential to the

ground, the microspines grip into the surface and establish a normal force, in this case, 710N (for the full palm seen in figure 1.5).

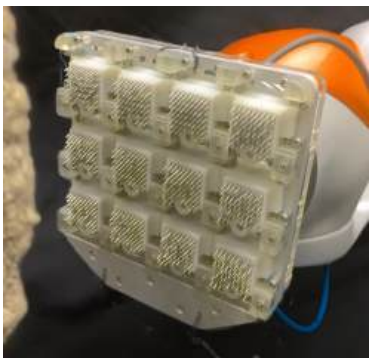

Figure 1.5: Microspine Palm. *Credit: Stanford - Evan Ackerman IEEE Spectrum 2016*

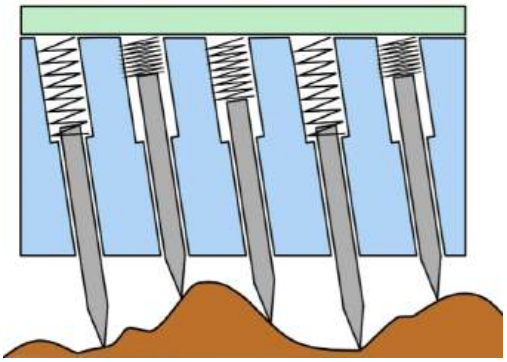

Figure 1.6: Cross section of a microspine tile. Spines and springs are inserted into angled holes with a bigger radius for the springs on top. *Credit: Stanford 2016*

Figure 1.6 shows the integration of spring-loaded spines into the tiles. This makes the spines compliant, such that they are able to adjust to the rough surface of rocks. The angle of the holes plays an important role in building up the required normal force for gripping, when sliding the gripper on the surface. Spine parameters, such as spine incident angle $\alpha$, as well as the spine tip radius $r_s$ is further explored in section 2.2.1.

### Example Gripper Integration - ReachBot, LORIS

When looking at the integration of gripping mechanisms into robots, individual use cases determine the effective configuration of the system. The grippers can be active or passive, for exploration on earth or in space.
One example of a robot exhibiting an active gripping mechanism is "ReachBot" [7], which was built to explore lavatubes and other inaccessible locations. The gripping mechanism is mounted on booms, which can extend to reach surrounding walls. The three arms on the gripper are tendon actuated and use different tendon routings for opening and closing, as seen in Figure 1.7. The same basic gripping principle as explained in "Example Microscale Gripping - Stanford Microspine Gripper" 1.2.1 is used, including compliant microspine tiles, that can move on a linear rail to be pulled along the surface.

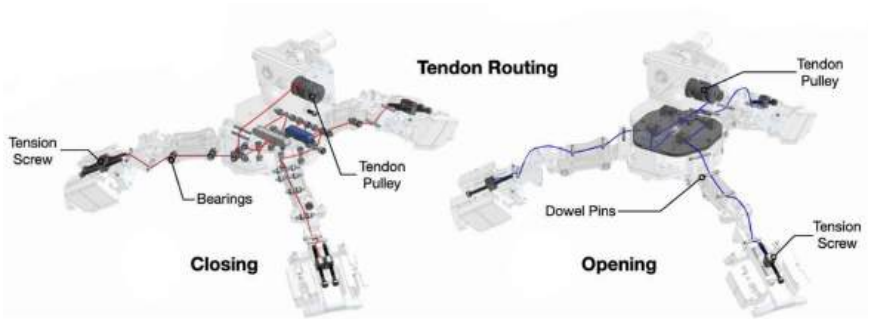


Figure 1.7: Tendon routing of ReachBot. *Credit: Q. Cheng et al., Science Robotics, 2024*

Another example of a climbing robot is "LORIS - A Lightweight Free-Climbing Robot for Extreme Terrain Exploration" [8]. The main difference in the gripping mechanism when compared to Reachbot, is that the gripper functions passively. The claw is comprised of microspines with built in compliance. When climbing on a rough surface, LORIS uses gravity to provide the neccessary downward force for the spines to grip into the surface.

Figure 1.8: LORIS scaling a rocky, vertical wall using passive microspine grippers. *Credit: Paul N. et al. LORIS: A Lightweight Free-Climbing Robot for Extreme Terrain Exploration 2024*

Table 1.1 shows a comparison of existing robots, which utilise spine gripping for climbing. As it is apparent, none of the robots are designed for zero gravity. Design parameters such as the incident spine angle are not listed here, as they are discussed in section 2.2.1. Furthermore, these robots are all using a static gait to climb.

| Robot | Terrain | DoF (per limb) | Mass [kg] | Gravitational Acc. [g] |
|---|---|---|---|---|
| RiSE [9] | Flat Walls | 2 | 3.8 | 1 |
| HubRobo [10] | Handholds | 3 | 3 | 0.38 |
| SCALER [11] | Handholds | 6 | 6.3 | 1 |
| LEMUR 2B [12] | Irregular | 3 | 10 | 1 |
| Lemur 3 [13] | Irregular | 7 | 35 | 0.38 |
| RockClimbo [14] | Irregular | 4 | 3.5 | 0.67 |
| LORIS [8] | Irregular | 3 | 3.2 | 1 |
| Reachbot [7] | Irregular | 5 | 11 | 0.38 |

Table 1.1: Comparison of existing robots used for climbing. *Credit: Adapted from [8]*

### 1.2.2 Gripper Body

For a gripper, a crucial part is the body where important things like the motor, tendons, the gripper linkage, and other parts like a differential mechanism are located. One main difference between the grippers is the way the motor is used. In the "spiny gripper" [14] and also the ReachBot [7] the motor acts as a driving force for both the closing and opening of the gripping legs as can be seen in Figure 1.9 and Figure 1.7. In another gripper [15] the motor only drives the detachment/opening of the gripping legs, and the attachment/closing is done through a pre-tensioned spring. The mechanism can be seen in Figure 1.10.

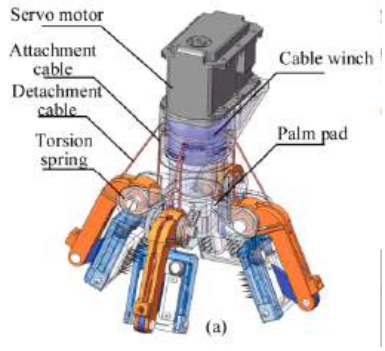


Figure 1.9: Spiny Gripper. *Credit: Peijin Zi et al. Mechanism and Machine Theory, 2023*

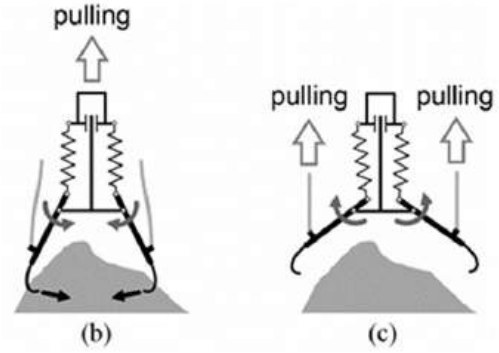


Figure 1.10: Passive Gripper. *Credit: K. Nagaoka et al., IEEE Robotics and Automation Letters 2018*

Most grippers have some sort of tendon mechanism for either closing, opening, or both. The tendons are then either connected directly to a pulley, driven by the motor [14, 15], or they are connected through a differential mechanism, which enables the tendons to have equal force distribution [5, 7, 16]. In the ReachBot they use a three-way whiffletree load-sharing mechanism, seen in Figure 1.11. The JPL Nautilus in Figure 1.12 gripper uses some form of a translating plate to which a continuous loop tendon is connected.

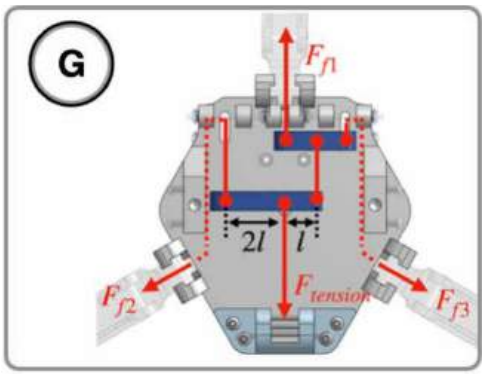


Figure 1.11: Three-way whiffletree load-sharing mechanism of ReachBot. *Credit: Q. Cheng et al., Science Robotics, 2024*

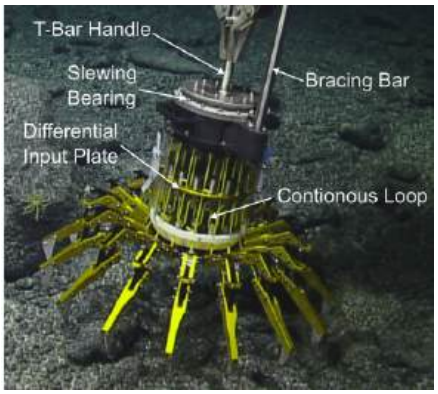


Figure 1.12: JPL Nautilus Differential: Contious cendon loop driven by input plate. *Credit: K. Nagaoka et al., IEEE Robotics and Automation Letters 2018*

### 1.2.3 Testing of Grippers

To evaluate the gripper's performance and capabilities of gripping various tests are conducted. Given that most grippers are designed with a hierarchical structure, tests were performed at multiple levels: individual microspines, single gripping legs, and the entire gripper assembly. The primary focus of these tests was to determine the attachment force that the gripper can generate.
In the literature, several studies have investigated the performance of single microspines. For instance, Q. Cheng et al., Acta Astronautica, 2024 conducted adhesion tests of a single linkage with a single mircospine. For the test, they pulled the tension rope to move the linkage tangential to the rock, seen in Figure 1.13. After a sliding process the microspine attaches to the surface and begins to generate a force $F_x$ in tangential x-direction as also a force $F_z$ in normal z-direction. The maximum force $F_z$ is of interest as it indicates the peak normal force that the microspine can generate, which is crucial for evaluating its gripping performance and stability.


Figure 1.13: Single Microspine Testsetup. *Credit: Q. Cheng et al., Acta Astronautica, 2024*

For the measured forces of the test, one can see in Figure 1.14, that for a generated force $F_x$ of $77N$ a pulling force $F_z$ of $22N$ is possible. The graph shows the sliding and adhesion process, as also the recovery and resting process. In another test of a different gripper [16] can be seen in Figure 1.15. There we can see for a single cassette the force and displacement measurements. As the linkage is pulled in x-direction and later also in z-direction, the slipping and catching moments can clearly be located where there's a significant drop in the forces. For this test a pulling force $F_z$ of $3.5N$ is possible for a tangential force $F_x$ of $16N$.

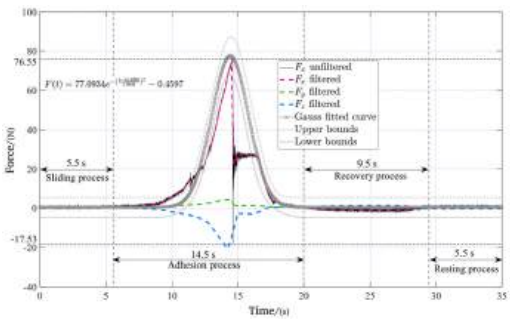


Figure 1.14: Single Microspine Measurement Plot 1. *Credit: Q. Cheng et al., Acta Astronautica, 2024*

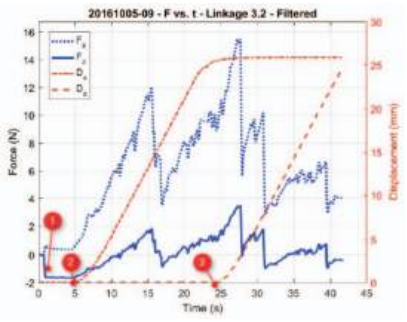


Figure 1.15: Single Microspine Measurement Plot 2. *Credit: A. Parness et al., IEEE Aerospace Conference, 2017*

For the entire gripper, P. Zi et al. [14], conducted a series of tests to evaluate the

capabilities of the gripper to adhere to different types of rock surfaces. The tests were conducted on porous basaltic rock. After the gripper was placed on the rock, the gripping sequence would be initiated and after adhesion, the gripper would then be pulled normally and tangentially w.r.t. the surface, as can be seen in Figure 1.16.

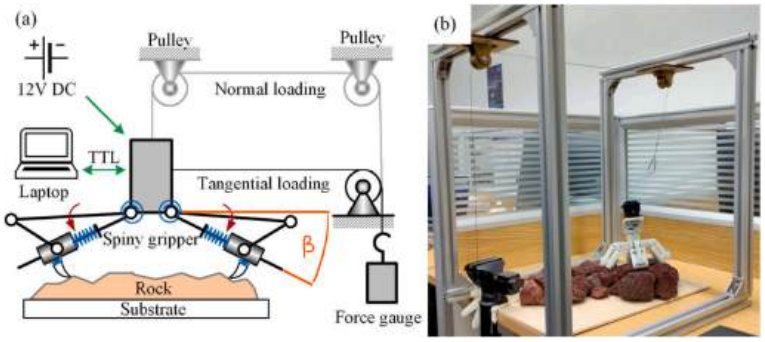


Figure 1.16: Example test setup for a microspine-based gripper. *Credit: P. Zi et al., Mechanism and Machine Theory, 2023*

The forces were measured with a force sensor and for different deployment angles $\beta$ of the gripping legs. In Figure 1.17 the resulted forces of the normal and tangential loading are plotted for the different $\beta$ angles. For normal loading a maximum pulling force could be generated of $39.7N$ with an angle $\beta_{average}$ of the gripping legs of 35°. For tangential loading the optimal angle $\beta_{average}$ was 25° with a maximum pulling force of $47.5N$.

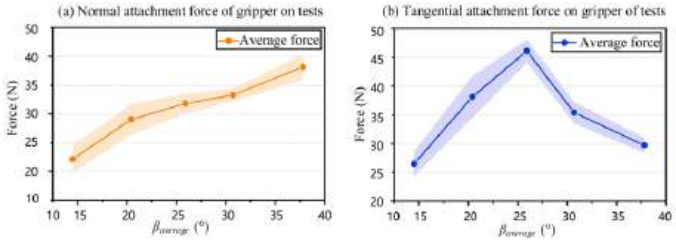


Figure 1.17: Test results for the spiny gripper. *Credit: P. Zi et al., Mechanism and Machine Theory, 2023*

In the scope of our project, we want to design a gripper which can function in multiple gravity setups, including zero gravity. Moreover, the gripper shall be designed to work in a dynamic scenario, specifically to grip upon impacting the surface with the robot, which, so far, has not been demonstrated yet.

## 1.3 Vision

The goal of this bachelor thesis is to develop a functioning prototype of a robotic gripper, designed for applications in microgravity. The focus will hereby lie on the design and testing of an underactuated gripping mechanism. The design of the robotic gripping mechanism consists of multiple sub-goals, which are described below.

1. Development of a concept for a robotic gripping mechanism, including project-relevant design specifications, as elaborated in 2.1.
2. Realization of a full CAD model depicting the robotic gripper with all its parts.
3. Integrating an actuation system capable of initiating the attaching and detaching sequence.
4. Testing the grippers capabilities of attaching to various types of rocks, including the tests of the microspine block and the hook mechanism.
5. Integration of the gripper into SpaceHopper (SH), a former focus project affiliated with ETH Robotic Systems Lab (RSL).
6. Design of the following adapters for the system, such that functionality tests may be performed.
    - Design of an adapter to fit the gripper onto the Franka robotic arm, which will be used for testing at ETH.
    - Design of an adapter for SpaceHopper to fit onto the ESTEC platform *MANTA*.
7. Integration of an impact-dampening system with the ability to trigger the gripping sequence upon contact with the surface.

The system definition and specifications are further explained in the section “System Design” 2.1.

# Chapter 2

# Concept

This chapter aims to give a detailed overview of the design of the gripper. For this purpose, firstly, the system requirements and system identification are being discussed. Before introducing the final design, scrapped designs are evaluated to give an insight into the thought process during the concept phase.

## 2.1 System Overview

### 2.1.1 System Requirements

The use of any system in space presents additional challenges to make it suitable for harsh environments, encompassing a wide range of temperatures, surface and weather conditions. In this project, the design will address these challenges in a general manner, without focusing on a specific scenario. This means that instead of using a case study, the design aims to be usable in various environments, including different gravitational accelerations. However, SpaceHopper is chosen as an example robot into which the gripper could be integrated. The general system requirements are listed below.

- The system shall be designed around extreme cases, setting the lower bound for the gravitational acceleration at $0ms^{-2}$ and the upper bound at $8.87ms^{-2}$ (gravity on Venus). These values were chosen, because $8.87ms^{-2}$ is the next lower gravity found on an extra-terrestrial object in the solar system, after earth's $9.81ms^{-2}$.
- The system shall be underactuated, to keep the actuation control simple and the weight light. This is important, since a simple design allows for less error sources.
- The design shall provide a combination of mm scale and macroscale gripping, such that it is suitable for terrain with hierarchal features.
- The design shall represent a module suited for integration to robots.

To quantify point 1 of the system requirements above, the required pulling force of the grippers were plotted against gravitational acceleration. The scenario in figure 2.1 shows the required pulling force, if SpaceHopper would hang upside down on a wall with all legs and gripping legs attached to the surface. The weight of SpaceHopper is assumed to be 7kg, the weight of one gripper is 0.45kg. The complete MATLAB script can be found in Appendix A.1.
It can be seen, that for Venus, a minimum gripping force of 8.18N per gripper leg is required.

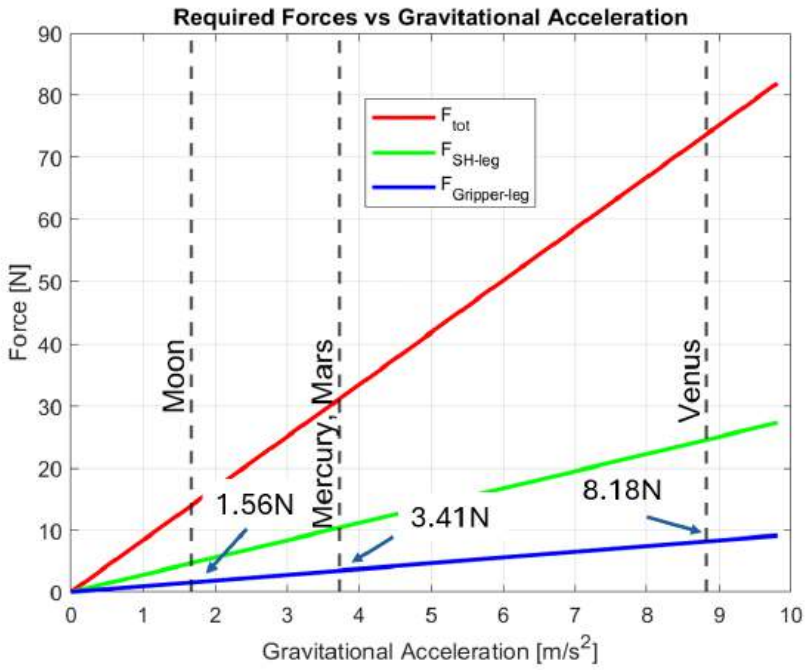


Figure 2.1: Required gripping force Of SpaceHopper hanging uside down in dependency of gravitational accelerations $[ms^{-2}]$ of bodies in our solar system. In red: Total required pulling force. In green: Required pulling force of one SH leg, respectively one gripper. In blue: Required pulling force of one gripper leg.

Through the possible integration into SpaceHopper or other robots, the system can be tested in a dynamic test scenario, specifically, jumping from a surface to another. Therefore, the system has to be laid out for impact dynamics. In zero gravity, this means that the impact has to be heavily damped, such that the robot does not bounce back into space. For the jumping scenario, further specifications to the design are listed below.

- Reliable static gripping on various relevant surfaces, such that SpaceHopper is safely attached to the surface it jumps from.
- Sufficient shock absorption during dynamic contact with the surface, such that the structural integrity of all parts in ensured.
- Ability to trigger the grasping sequence, once contact with the surface is established.

### 2.1.2 Subsystem Identification

The system can be divided into mechanical and electrical sub-systems. Electrical components are sensors, electric actuation and power supply, whereas the mechanical sub-systems are the tendon-driven actuation, linkage and the gripper foot. The gripper foot describes the end-effector of the gripping mechanism, which is responsible for the gripping of features on rock surfaces 2.2. The linkage 2.3 connects the gripping foot to the housing of the gripper, which holds the actuation mechanism and possibly the power supply.

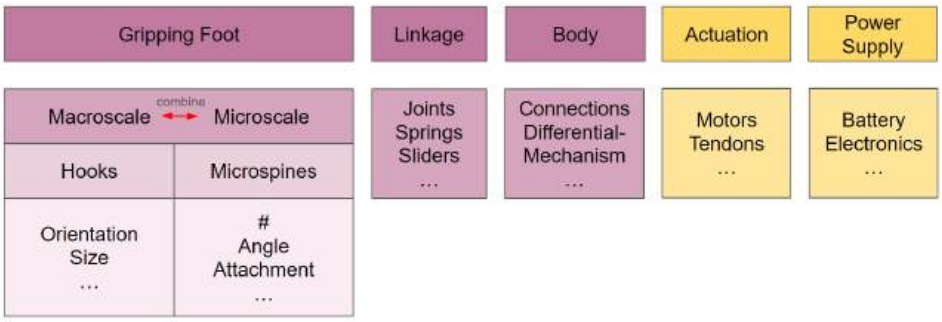


Figure 2.2: Sub System Schematic. Mechanical sub-systems in purple, electronic sub-systems in yellow.

## 2.2 Gripping Foot Design

This section aims to give insight into the design process, underlying theory and the final design.

### 2.2.1 Spine Parameters

Microspines are widely used in gripping robots such as RiSE [9] and SCALER [11]. To exploit the gripping technology to its fullest potential, some parameters have to be taken into account when designing a microspine foot. The following section concentrates on the spine incident angle, spine compliance, as well as dependencies between incident spine angle, gripping force and amount of spines per foot.

#### Gripping Force

Gripping of a spine can be achieved through two main mechanisms. The first mechanism is based on friction, seen in Figure 2.3 on the left. Pushing the spine in tangential direction w.r.t. the surface, creating a force $F_n$, a pulling force $F_f$ can be generated. The bigger the tangential force is, the bigger the pulling force gets. The second mechanism, seen in Figure 2.3 on the right, is based on the geometric constraint the surface implies on the spine. A tangential force is needed to push the spine in the asperity. With this, a pulling force also can be generated.

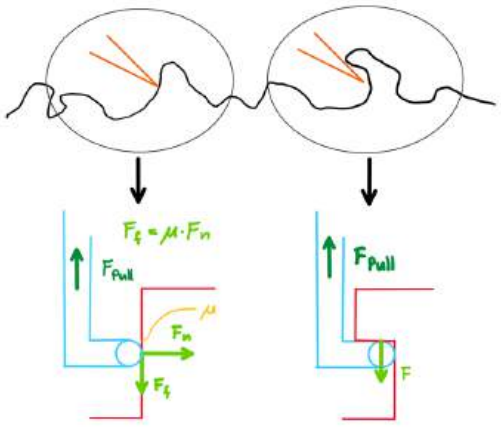


Figure 2.3: The two main hooking mechanism for a spine

In both cases a tangential force is needed to create a pulling force. The maximal pulling force is then limited by the applied tangential force, the strength and structure of the surface, and its friction coefficient.

### Spine Parameters

When used in a certain environment, the gripping foot design depends on the local surface conditions such as roughness of the rock, or other environmental factors, for example dust.

One important aspect given by the environment, is the rock's asperity size, meaning, the size of holes in the surface, which the spine can grip into, and therefore, attach to. The asperity size determines the spine tip radius $r_s$. Assuming a fractal surface, the number of asperities decreases with $\frac{1}{r_s}$ [17]. However, since providing a detailed scan of the test rocks was out of the scope of this project, the spine tip radius was neglected in the design of the microspines.

Another important factor for gripping is the incident spine angle $\alpha$. This is because together with $r_s$, $\alpha$ determines the number of asperities the spine can reach. In existing literature, a variety of incident spine angles are installed in the mechanisms, due to varying use cases. This is why in this project, $\alpha$ is determined experimentally. For further information about the test, refer to section 3.3.
Nevertheless, as described in [18], some general dependencies can be followed, when designing the gripper:

$$n \propto \frac{1}{r_s} \tag{2.1}$$

$$n \propto m \tag{2.2}$$

$$r_s \propto \textit{targert asperity size} \tag{2.3}$$

$$\textit{spine strength} \propto r^2 [17] \tag{2.4}$$

where $n$ is the number of spines and $m$ is the mass of the gripper, respectively the robot.

### Spine Compliance

The need for compliant elements in the design becomes apparent, when looking at the gripping sequence.
First, the gripping foot gets lowered down onto the surface. Assuming the foot passively orients itself parallel to the surface, the spines are contacting the ground. However, since the surface is assumed to not be perfectly level, only some spines will contact the ground. Therefore, compliance in z-direction (normal to the surface) is required. This can be achieved, by installing springs in the system, as is shown in figure 1.6. The stiffness of the springs should be low enough, to allow for passive adaptation of the spines to the surface.
In a second step, the gripper gets pulled parallel to the surface. Here, compliance in x-direction (parallel to the surface) is needed, to ensure that spines, which have not yet gripped into the surface still have the translational freedom to do so, while other spines are already attached. In the microspine tile design, as shown in figure 1.6, this is not possible. A possible design employing normal and translational compliance is demonstrated in the design of the robot "SPINYBOT" [17], seen in figure 2.4. Compliance is integrated into the design, by using soft polymers acting as spings inbetween spines.


Figure 2.4: SPINYBOT's compliant microspines. Normal and translational compliance is integrated by using soft polymers acting as springs in between spines. *Credit: Alan T. et al. Scaling hard vertical surfaces with compliant microspine arrays. The International Journal of Robotics Research, 25:1165– 1179, 2006.*

### Spine Configuration

Figure 2.5 shows research [7] explaining the correlation of hook/spine design and the amount of asperities reached. The higher the amount of asperities that a gripper can reach, the more likely it is that the spines will catch onto it. Looking at figure 2.5, one can see that case 5 represents the most ideal design. It is composed of a spine connected to a ball joint, which also has translational freedom. This is similar to a rotating hook, which is simultaneously able to slide. Case one on the other hand can be compared to a spine with compliance in z-direction. Combining these features addresses the goal of combining macro feature gripping, i.e. the hook, with microfeature gripping, i.e. the microspines. This suggests that the implementation

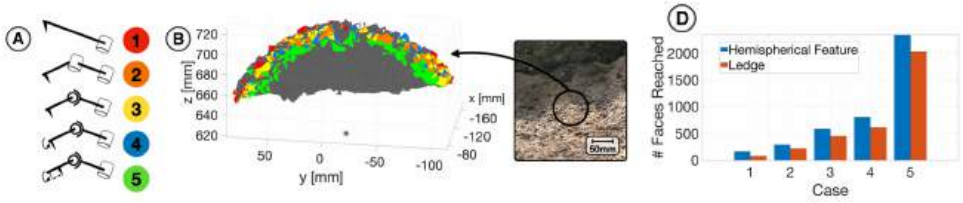


Figure 2.5: A: Comparison of reachable areas on rock surface for different spine orientation cases 1-5. D: #Faces reached for cases 1-5 *Credit: Tony G. C. et al. Locomotion as manipulation with ReachBot. Science Robotics Research Article 2024*

of a hook into the system would be a valuable addition to the gripper. Section 2.2.2 explores how a combination of hooks and spines might be implemented.

## 2.2.2 Design Iterations

The foot's design process started with the goal of combining micro- and macrofeatures in the surface of rocks. This stands in contrast to existing grippers, which focus on only one of these scales. For example, JPL's nautilus gripper [5] focuses on gripping into bigger, softer rock features, whereas Stanford's microspine gripper [6] is purely applicable to mostly planar surfaces with microscale features. To combine these features, the aim is to design a foot which grips simultaneously with a bigger hook and microspines. During the design phase, multiple concepts were evaluated. Figures 2.6 to 2.8 show sketches of some of the design ideas that were discarded for different reasons. Figure 2.6 shows a gripper that is inspired by hooks used for mountain climbing. While this design included features able to grip on macro and microscale level, it does not prove useful when gripping on convex or concave terrain due to lack of compliance. Another idea was to span a net-like structure around the body of the gripper which would pull towards the body when actuated, similar

to the structure shown in figure 2.7. This design idea was inspired by a ducks foot, which has a membrane between each toe. The motivation to include this concept was that the increased contact area might pose an advantage when gripping on a sandy surface. However, the non-rigidity of this design makes it difficult to place hooks or spines along the net and to ensure gripping. Further, in case of entanglement of the net on the surface, the opening of the grasp would not be guaranteed.

Other designs had to be discarded because of their dependency on gravity. This especially included designs that comply to the surface structure, when a sufficient normal force is applied to the structure. In the case of this project, the design is required to function in zero-gravity environments, which makes the application and maintenance of such a normal force impossible without pushing the robot back into space. This type of mechanism would work if it is assumed that at least one limb of the robot is already attached to the surface. However, this gripper is meant to be implemented into SpaceHopper, a robot designed for jumping from surface to surface, rather than using a static gait. Hence, the gripping sequence has to work when the robot approaches a surface without prior attachment of a limb.


Figure 2.6: Gripping foot inspired by mountain climbing hooks.

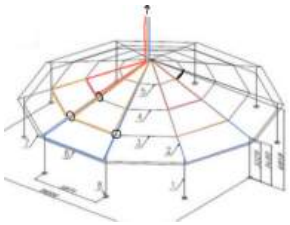

Figure 2.7: Basis for a net inspired design. Each colour indicates a single tendon. *Credit: S. Pellegrino, copyright 2001 by Springer-Verlag Wien*

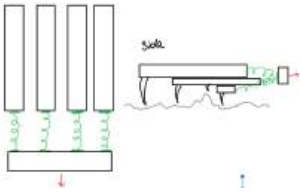

Figure 2.8: Gripping foot with an array of different sized spines.

As inspiration for the gripping foot, an eagle claw was observed. As seen in figure 2.9, the foot consists of multiple toes with a heel and claw. On the right side, figure 2.10 shows a preliminary sketch how the heel and claw could be implemented with microspines and a hook. The following subsection "Hook mechanism" 2.2.2 will go into further detail on the possible implementation of hooks into a body housing microspines.

### Hook Mechanism

When implementing a hook in addition to microspines, multiple configurations are possible. As seen in figure 2.5, a rotating and simultaneously translating hook can reach the highest number of asperities. In this project a rotating hook, a sliding hook and a rotating and simultaneously sliding hook were compared. A test on the established gripping force showed that the rotating plus sliding hook can reach the highest gripping force, followed by the rotating hook, both showing a significant increase in force when compared to just a microspine body gripping. More information on the test can be found in section 3.5.
Figure 2.11 shows, how the gripper could be implemented into SpaceHopper. On each of SpaceHopper's legs sits one gripper. The gripper is composed of the motor

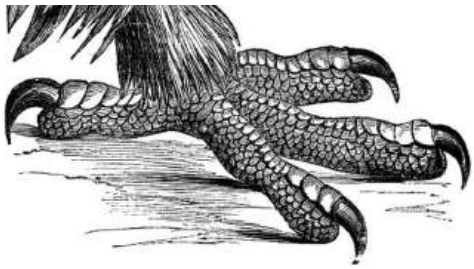

Figure 2.9: Drawing of an eagle's foot touching the surface. *Credit: iStock*

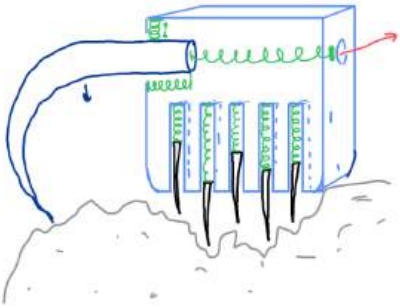

Figure 2.10: Eagle claw inspired design, combining microspines with a macroscale hook

housing and three legs which are comprised of Linkages and a gripping foot. The right side of the picture shows part of a gripping foot, namely, the spine body with an insert (in green) for a ball bearing and hook holder (in pink).

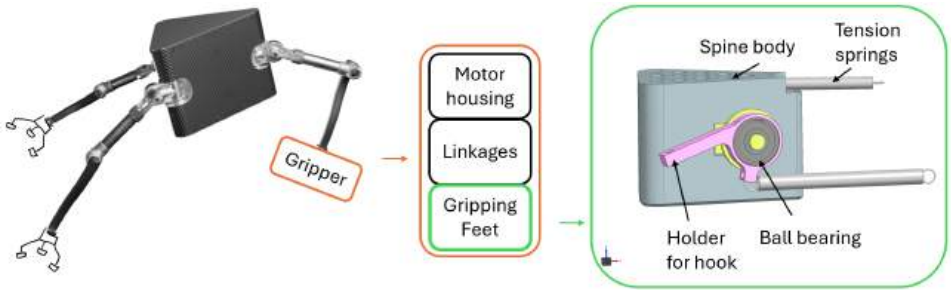


Figure 2.11: Schematic of how the gripper could be implemented into SpaceHopper. The gripper on SpaceHopper's leg consists of the motor housing, the linkage and the gripping foot. The right hand side shows the CAD of a spine body (belonging to the gripping foot) with an insert (green) to fit a bearing and rotating hook holder (pink).

The insert sits in a fitting hole in the spine body. Depending on the hook configuration, the hole can be extended to a slider. In the test described in section 3.5, it was found that a sliding and rotating hook produces the highest pulling force, namely $5.35N$.

When designing the slider, St. Venant's principle was used to dimension the parts, such that they would not jam, which is elaborated in appendix A. Even though the sliding plus rotating hook resulted in a higher pulling force compared to the $5.15N$ from the rotating hook, the standard deviation proved to be much higher. This is why, if a hook was to be implemented, a purely rotating hook should be chosen. Section "Differential Design" 2.3.1 explains, why the implementation of a hook into the gripping foot was discarded in the end.

### 2.2.3 Final Gripping Foot Design

For the final gripping foot design, a block holding 4x4 spines was chosen. Due to the easy integration and availability, instead of microspines, 12mm long nails with

a diameter of 1mm were chosen. A cross section of the spine body can be seen in figure 2.12. By testing the gripping force of a block with 9 microspines and $5N$ of tangential pulling force for different incident spine angles, an optimal angle of 50° with respect to the ground was found. The movement of the nails is constrained by springs, which are inserted into slightly bigger holes on top of the nails. The springs have a diameter of $2mm$, a length of $10mm$ and a spring constant of $0.192N(mm)^{-1}$. The spring constant was chosen in accordance to the availability of springs of the right size. It was chosen to be as small as possible, while not equaling zero, such that the nails can easily retract into the body when complying to the surface and are brought back to their initial position by the springs after disengagement. To contain the springs and nails, a lid can be screwed on top of the spine body.

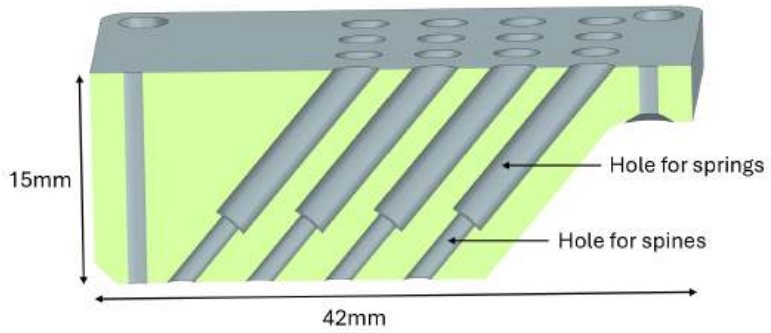


Figure 2.12: Cross section of the spine body. the whole in which the nails and springs get inserted are angled 50° wrt. the ground.

The dimensions of the parts were mostly determined by the restrictions of the 3D printer. A hole size of 1.4mm for the spines was found to be the minimum feasible diameter that was reliably reproducible with the available 3D printers.

## 2.3 Linkage Design

The linkage design was directed by two major requirements. First, the foot of the gripper has to be parallel on the surface, regardless of the orientation of the robot's body. Secondly, the foot has to be pulled inwards parallel to the surface during the entirety of the closing phase. To incorporate these requirements into the design, two concepts were developed, one with only linkages and one with a combination of linkages and a tendon.
This section explains how the requirements are incorporated into the designs, as well as the designs' advantages and disadvantages.

### 2.3.1 Design Iterations

#### Tendon Design

Figure 2.13 shows the CAD assembly of the tendon design.

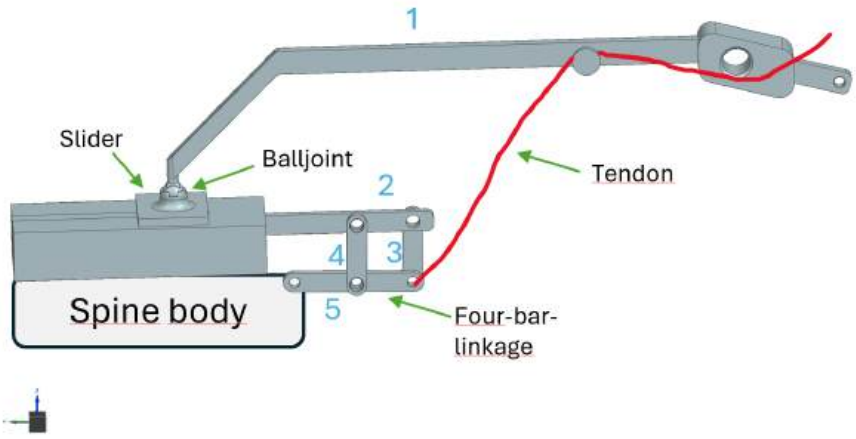


Figure 2.13: Schematic of a tendon driven mechanism. The tendon can be seen in red.

To passively orient the gripping foot on the surface, a link with the male part of a balljoint is joined to the female counterpart on top of the foot. The balljoint is initially placed centrally on the foot. The female part of the joint is part of a sliding mechanism.

The parallel direction of pulling on the spine body wrt. the ground is achieved by integrating a four bar linkage into the design of the gripping foot. A tendon is routed from the motor over link 1 to pull on the connecting rod of the four bar linkage. Link 2 represents the fixed rod of a four-bar linkage with equal length to the connecting link (link 5) at the bottom. By making these equal length and because of the limited range of translational travel the differential has to achieve, the motion of the connecting rod can be assumed as linear, which fulfills the requirement for small angles.
However, once the spines attach to the ground or the maximum travel of the linkage is reached, the pulling tendon will exert a force causing a moment on the foot, which would cause it to lift from the ground. Hence, this design was discarded. Another reason to abandon the design was the worry that the tendons might become loose or float around in space or get caught on surface features.

### Differential Design

As discussed in the subsection "Hook Mechanism" 2.2.2, different hook configurations are possible. When implementing the sliding plus rotating hook, it has to be emphasized, that the hook and spines should act complimentary to one another. This means, that when the spines are gripping, the hook should still have enough translational freedom to find an asperity to grasp on, and vice versa. To achieve this, a differential pulling mechanism is introduced.

Figure 2.14 shows the CAD assembly of the differential design. It consists of a spine body with an insert for the hook configurations, as explained in 2.2.2. It has to be noted, that in comparison to figure 2.11, figure 2.14 shows the configuration of a sliding plus rotating hook. The body is closed off with a lid (orange), which contains holders (blue) for the sliding shafts (red).

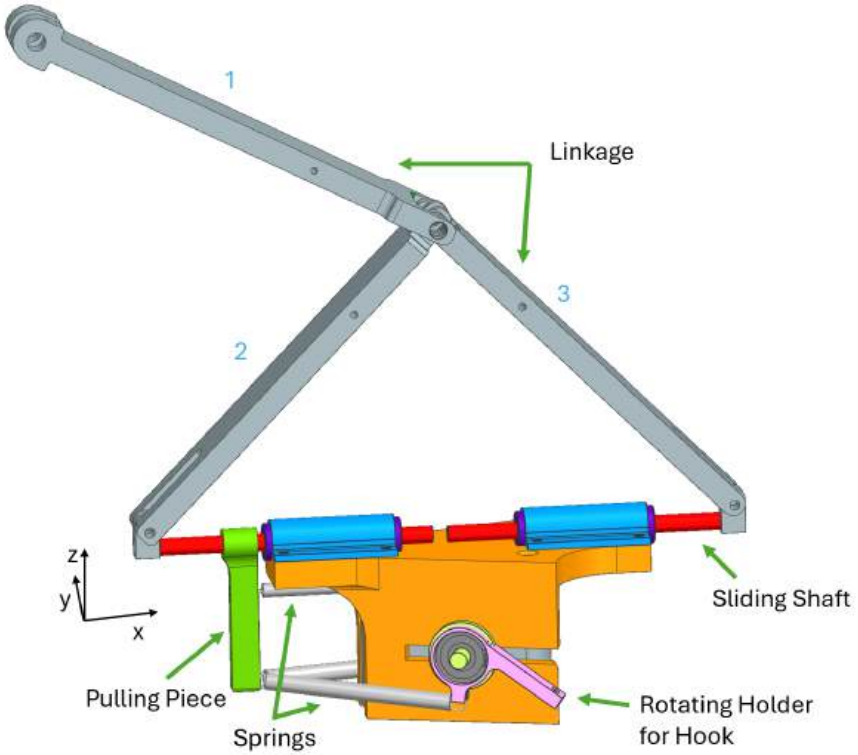


Figure 2.14: Complete differential design

The term "differential", here refers to the fact that the hook and the spine body can be moved independently from another, if the hook is implemented as a sliding, or sliding plus rotating hook. This is achieved by connecting the hooks and the body with separate springs to a connection piece (green), which gets pulled with the shaft.

Here, the foot's orientation is achieved by implementing two rotational DOFs in the linkage. Starting from the motor housing, link 1 rotates downward, pushing link 2 and 3 down and out. This linkage connection represents the rotational DOF around the y-axis. Link 2 and 3 are themselves connected to two sliding rods, which are constrained to move linearly in x, but free to rotate around the x-axis.

The parallel pulling is ensured from the fact, that the differential is pulled by a piece which is connected to the linear sliding rods. This axis of the rod itself is con-

strained parallel to the top of the gripper by the bushings (violet) and the holders (green). Because the pulling part is perpendicular to the slider, the differential is always pulled parallel to the shaft and therefore to the rocks surface.

However, this design has some issues. While the sliding mechanism establishes a gripping force through the spines, once a force is applied in a direction other than the tangential pulling force, the gripping might fail. These forces could for example stem from the weight of the robot while walking. This would pose a force on the motor housing of the gripper and therefore on the links of the gripping mechanism, causing them to move, since it is free to slide through the shafts. It is necessary to remember that the force established by the spines is transmitted through the force of the spring which is connected to the sliding shaft. If the shaft (which gets pulled by link 2 and 3) is set in motion, the spring loses its pulling force and therefore the gripping strength can not be ensured. Recall, that

$$F_s = k\Delta x. \tag{2.5}$$

Therefore, the conclusion is that this double-sliding design is suitable for zero-$g$ and possibly for microgravity, but not for environments with significant gravitational accelerations. Additionally, the design is very complex. The multiple sliding mechanisms in combination with the differential pulling proposes a solution that is prone to errors. This is why the design was discarded. Especially because of the low quality of the 3D prints, the concept of a sliding plus rotating hook was not successful. Instead, a simplified version was implemented, as can be seen in the following.

**Simplified Linkage Design**

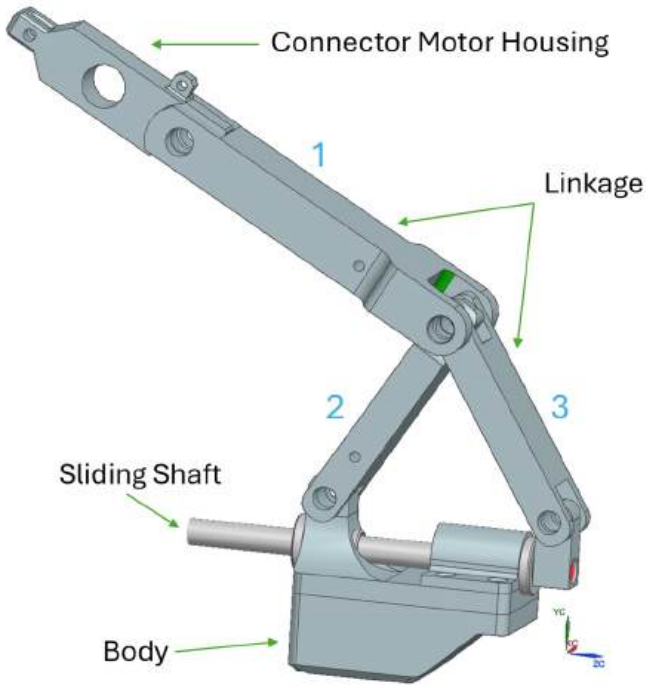


Figure 2.15: Simplified design

To simplify the mechanism, the differential pulling solution and the slider were discarded. As can be seen in figure 2.15, the mechanism still consists of the spine body, a sliding shaft and the linkage. The dimensioning of the design was decided by two parameters. The force transmission and the distance, the spine body is able to travel over the surface in order to grip ($\Delta x$). In the spine angle test described in section 3.3, the average normal pulling force of $3.5N$ was achieved with 5N of translational force. Remembering the requirement of $8.18N$ normal pulling force per gripper leg, this is clearly not enough. Hence, the translational pulling force was doubled from $5N$ to $10N$. As mentioned in 2.2.1, the number of spines scales with the mass that is to be pulled. This is why in this design, the number of spines was increased from 9 spines in the test, to 16. By calculating the force transmission from the motor to the linkage and setting the travel on the surface to $20mm$, the linkage dimensions could be determined. The calculations can be found in Appendix A.3.

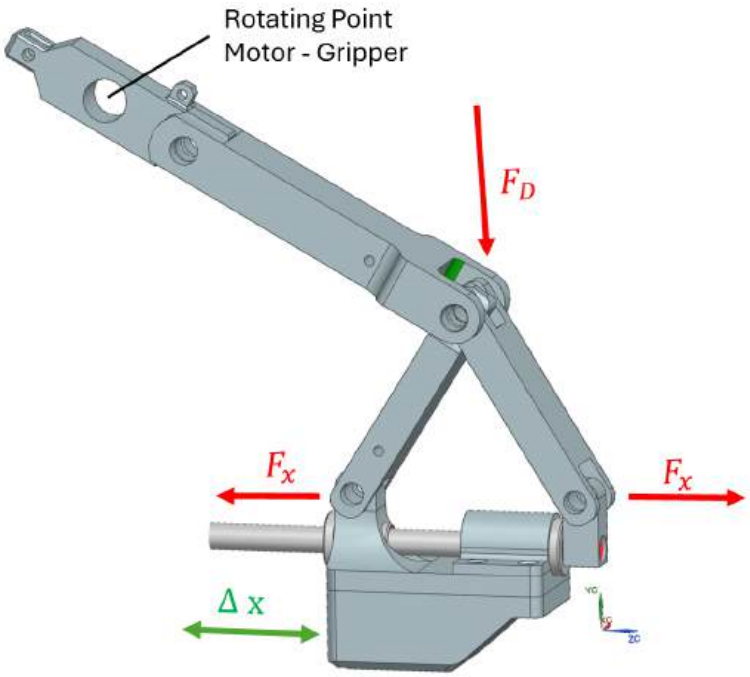


Figure 2.16: Simplified design with forces acting on the mechanism.

Figure 2.16 shows the forces acting on the mechanism. In this case, $F_x = 10N$ and $F_D = 33N$, creating a force ratio of $\frac{F_x}{F_D}$. If a force ratio of 1 and a travel of $10mm$ were to be achieved, the dimensions of the linkage would get much bigger, making the design too bulky to implement it. In figure 2.17 and 2.18, a size comparison can be seen.
Even though a bigger gripping force is to be expected out of the bigger design shown in figure 2.18 because of its many spines and force ratio, it is not guaranteed that all of the spines catch onto the surface. In fact, because of its bulkiness, it might even be expected that the spine body might not touch the surface in concave rock features and only a small area on convex surfaces. This is why the smaller design was chosen to move forward with testing.
While testing however, the design failed due to the fact that when link 1 rotates down, instead of creating a gripping force, the reaction force in the rotating joint caused the motor housing to get pushed upwards and no gripping force was estab-

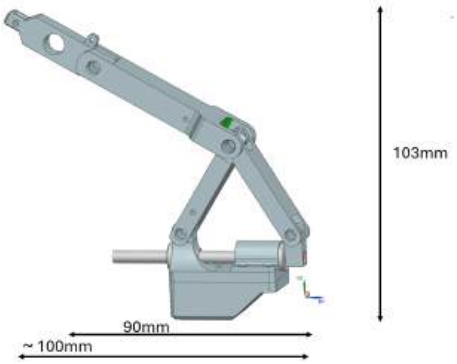


Figure 2.17: Measurements of mechanism with a force ratio of 0.3 and travel distance of $20mm$.

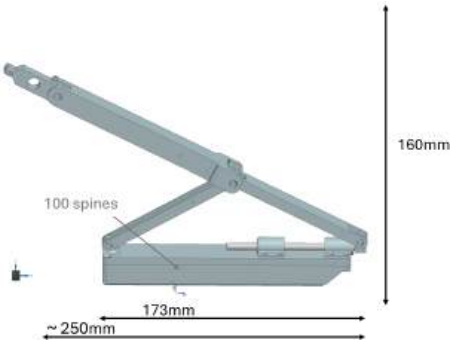


Figure 2.18: Measurements of mechanism with a force ratio of 1 and travel distance of $10mm$.

lished. This is why even though this design fulfills the requirements of simplicity, adjustment to the surface and pulling parallel to the surface, it had to be discarded.

### 2.3.2 Final Linkage Design

To be able to prevent the upwards movement of the motor housing, the rotation of the link 1 about the point marked in figure 2.19 has to be limited.

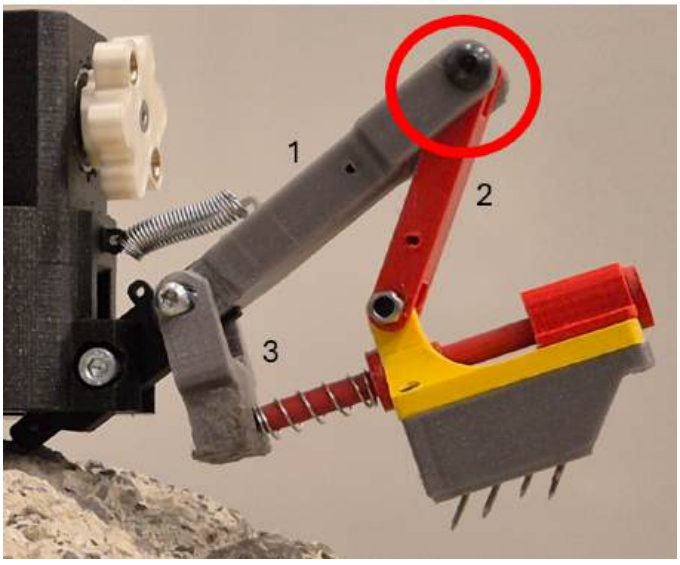


Figure 2.19: Final linkage design

To achieve this, the sliding rod is rigidly attached to link 3, which is free to rotate. This way, when link 1 rotates downwards, the gripping body slides inwards, pushed by link 2. Any upwards motion of the motor housing now drives the spines further along the rock, increasing the gripping force. The maximum gripping force is established, when the motor cannot drive link 1 further down, because the spines have gripped into the surface. By installing a spring between the spine body and link 3, the gripper gets passively released into its initial position when link 1 gets released by the motor.

## 2.4 Design Gripper Body and Pulling Mechanism

### 2.4.1 Design Gripper Body

The linkage with the gripping foot has to be somehow connected to a body capable of driving and actuating it. For this, a similar design was chosen as seen in P. Zi et al., Mechanism and Machine Theory, 2023 [14]. The whole gripper is designed modular and in a way to be easily mounted to feet of various robot legs, such like SpaceHopper.

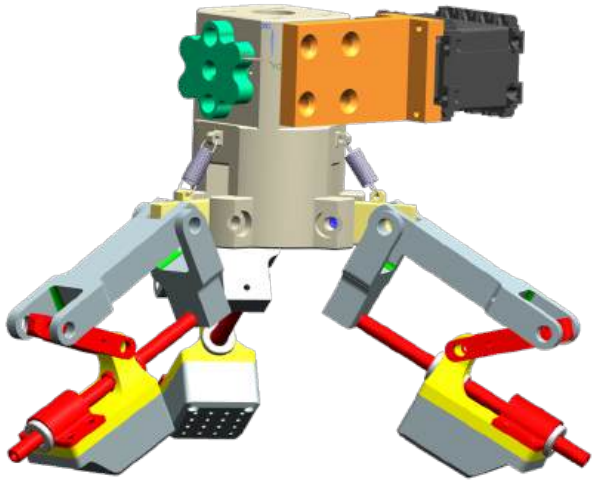

Figure 2.20: CAD of whole Gripper

An active gripping mechanism was chosen to allow direct control over the gripping force using a motor seen in Figure 2.21. Springs are connected between the body and the gripping leg to release the mechanism when the motor is disengaged passively.

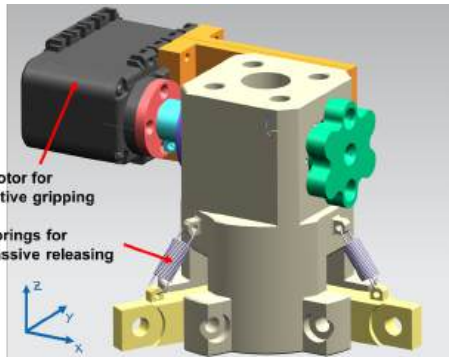


Figure 2.21: CAD of Gripper Body

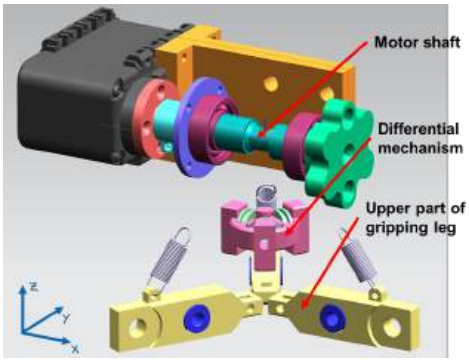


Figure 2.22: CAD of Body inside

A shaft is connected to the motor and a differential mechanism part with three pulleys is located inside the body.

### 2.4.2 Differential Pulling Mechanism

The gripper is under-actuated and driven by only one motor. The motor is mounted to a shaft on which a tendon is fixated. Through the rotation of the shaft, the tendon can get rolled up and thus the rotational motion is converted into a translation. The tendon could then be fixated directly to the gripper leg and this could be done for every leg. One drawback would be the force transmission of this system. Having connected the three gripper legs with tendons directly to the motor shaft has the serious drawback that the forces would not be equally distributed over the legs. When the shaft turns and the tendons get rolled up, all legs get deployed at the same time as desired, but if one of the legs gets hooked, the other legs can't be moved further. To address this problem a differential tendon mechanism, also used in the JPL Nautilus Gripper [5], is integrated in our gripper.

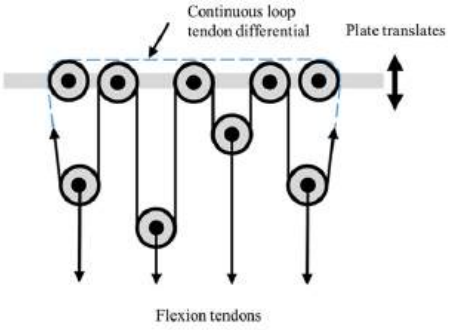


Figure 2.23: Differential tendon mechanism where on all the pulleys equal force gets applied, because of a single continuous tendon looped through the pulleys in an alternating fashion. The translating plate serves as the input and the flexion tendons as the output. *Credit: Backus et al. Design and testing of the JPL-Nautilus Gripper for deep-ocean geological sampling. J Field Robotics 2020*

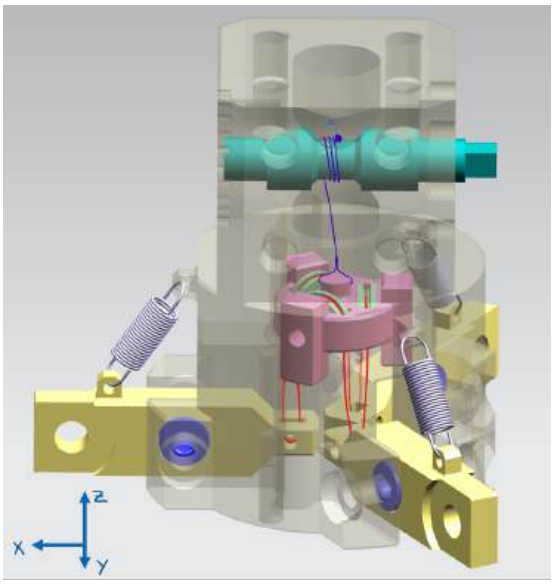


Figure 2.24: The differential tendon mechanism integrated into our gripper design

In Figure 2.24 one can see the differential mechanism integrated into our gripper

design. The shaft rolls up the blue tendon. “Dyneema” is the material for the blue tendon, because it has to withstand a lot of force. It is connected to the pink part, the differential mechanism part, which acts as the translating plate from Figure 2.23 and is free floating in the gripper body, only guided by rail guides. The pink part has this specific shape to keep it from jamming. The red tendon is a continuous loop and alternates between the pulleys on the differential part and the yellow gripper legs. For it, a nylon gut was selected and melted into a continuous loop. The force in the nylon tendon is 1/6th of the force in the ”Dyneema” tendon. Now when one leg has already hooked to the surface the other legs aren’t constrained and can continue to move further until they also latch onto the surface.

## 2.5 Actuation

For the actuation of the gripper, a DYNAMIXEL AX-12A (a servo motor) was selected. To control this motor the "DYNAMIXEL Shield MKR" was required. With this shield, a voltage of $9V$ - $12V$ can be supplied and an Arduino MKR zero can be connected. The Arduino MKR Zero then allows for easy control of the motor. The code used to deploy the gripper can be found in A.4.

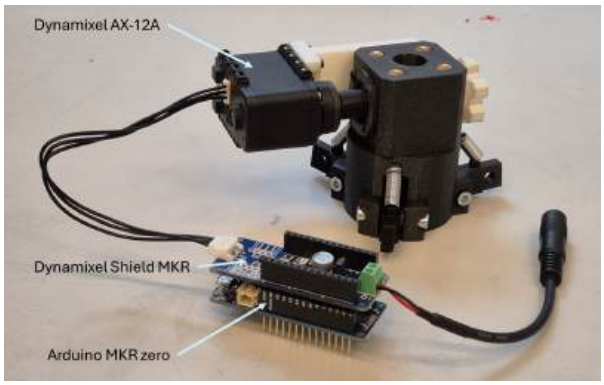


Figure 2.25: Connection of Motor with Shield and Arduino


Figure 2.26: Servo Motor: AX-12A

The motor can be operated in wheel mode and joint mode. With the latter, position control is possible with a running degree of 0° - 300°.

The servo motor has a maximum stall torque of $1.5Nm$. It also needs to be mentioned that this stall torque is the maximum instantaneous and static torque the motor can generate. Stable and exact motions are only possible until 1/5th of the stall torque ($0.3Nm$). While stable motions are not necessarily required for this application, the maximum applicable torque is expected to be considerably higher than $0.3Nm$. For the tests only 15% of the max. stall torque was used. With this torque of $0.1Nm$ a force of $30N$ is present in the first tendon. After the differential mechanism part, the force gets divided by three which leaves us with a force of $10N$ (because of the loop structure, the force in the second tendon is only $5N$).

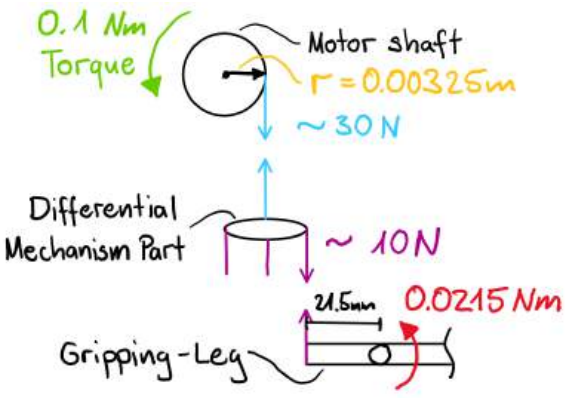


Figure 2.27: Present Forces in Gripper

With this mechanism, a final torque of $0.0215Nm$ can be applied to the gripping leg. Thus, the torque on the gripping leg is 0.215 the torque generated by the motor.

# Chapter 3

# Testing

A series of different tests have been conducted to validate the design choices and test the capability of the gripper to adhere to different types of rocks.

## 3.1 General Test Setup

All the tests were conducted with the help of a Franka Emika Panda robot arm [19] with a mounted 6-axis force-torque sensor. The robotic arm was chosen because of its versatility and its precision. Through commands, different position trajectories and desired forces can be generated for the end-effector. The code developed and written to control the arm and log the data can be found in the appendix A.5 and A.6. In the middle of the table, the different rocks get places.

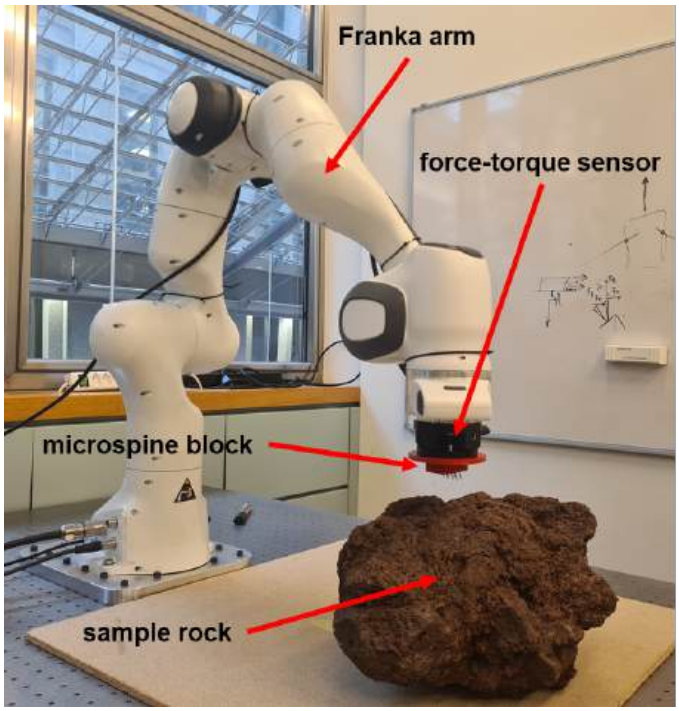


Figure 3.1: Test-setup with Franka arm, force-torque sensor and rock

## 3.2 Choice of Rocks

The tests were conducted on six different rocks to cover a various range of surfaces, that could resemble the topology or surface of asteroids and other planets or moons where the gripper could be deployed [14, 20, 21, 16]. Rocks with different roughnesses, different porosities, and overall varying surface structure were chosen. The chosen rocks can be seen in Figure 3.2

Figure 3.2: Rocks for testing: (top left to right) Asphalt / Tarmac, Limestone Tuff, Slate / Shale Stone (bottom left to right) Breccia, Volcanic Lava Rock, Sedimentary Rock

## 3.3 Microspine Angle Test

For the microspine angle test a block with nine microspines is mounted on the force-torque sensor on the Franka arm. The block is then manually placed on the test rock. Then the experiment is started by running a python file, which commands a force command and a position trajectory command. The end effector is pushed with a force $F$ tangential with approximately $5N$ in the x-direction while trying to follow a velocity $v$ in the normal z-direction as seen in Figure 3.3. The reference velocity is approximately $0.51\frac{mm}{s}$.

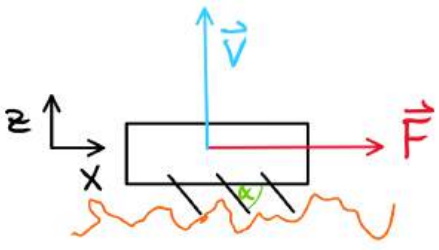


Figure 3.3: Microspine Angle Test Sketch

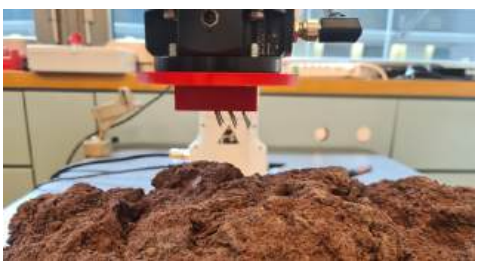

Figure 3.4: Microspine Angle Test

After the test starts, the position and forces in x, y and z are measured and logged to a file. For the evaluation, the forces are offset-corrected and then plotted.

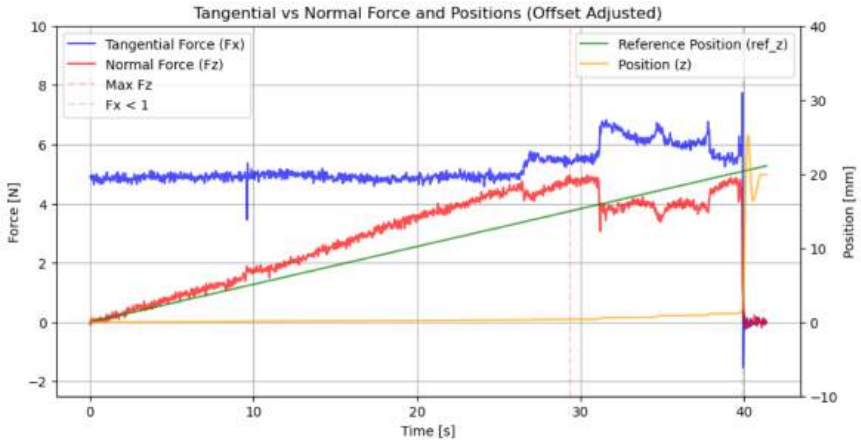


Figure 3.5: Sample Force Plot: tangential and normal Force

The plot in Figure 3.5 shows the commanded tangential force of approximately $5N$. One can also see the commanded reference position in green in z-direction and the actual z-position in orange. Through this discrepancy in the reference and actual positions, a force gets applied in normal z-direction plotted in red. After $40s$ one can see the detachment moment.
This test was conducted for six different angles $\alpha$ of the microspines relative to the block. The angles $\alpha$ for the microspines were: 90°, 80°, 70°, 60°, 50° and 40°. Per angle, the test was carried out for all six rock types, and 10 measurements were logged per rock.

## 3.4 Results for Microspine Angle Test

For the measurements, the maximum withstanding force in z-direction is the most elucidating metric. The bigger this force, the better the micro spine block can adhere to a surface and keep the gripper from detaching. With this, the maximum force in z-direction is located for all measurements and the average is calculated for each spine angle, as can be seen in Figure 3.6.

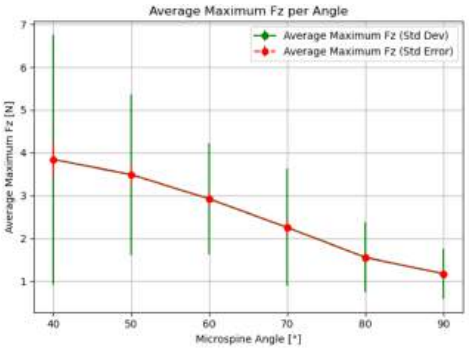


Figure 3.6: Average Maximum Fz per Angle. The green bars represent the standard deviation, and the red bars represent the standard error. The sample size for each angle is 60.

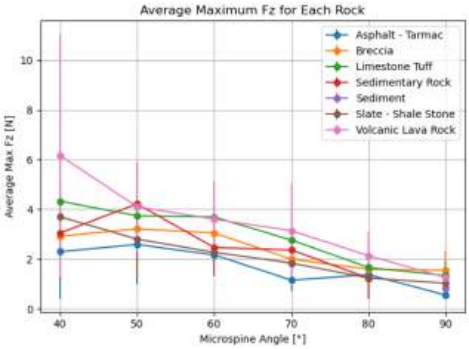


Figure 3.7: Average Maximum Fz per Angle for each Rock. Every color represents a rock type, and the bars represent the standard deviation. The sample size for each angle and rock is 10.

The bigger the spine angle the bigger the average maximum pulling force $F_z$. The reason for this is, that the smaller the spine angle, the better the spines can insert into the rock asperities and create a lock which can withstand higher pulling forces. Also worth mentioning is the standard deviation. For the 50° angle the standard deviation is considerably smaller than the one for the 40° angle. The gripper should exert the highest pulling force while also be es reliable as possible. This is the reason 50° was chosen for the spine angle in the final design. With this angle an average maximum pulling force $F_z$ of $3.49N$ can be generated, seen in Table 3.1. Because the whole gripper consists of three micropine blocks, the expected pulling force would be $10.47N$.
In Figure 3.7 one can see the average maximum pulling force $F_z$ per angle plotted individually for each rock type. For rock types with higher porosity, a rougher surface, and an overall less flat topology, the pulling force $F_z$ is on average bigger than on rocks with a smoother and less curved surface. But overall the microspine can generate pulling forces on all tested rocks.

| **Spine Angle [°]** | **Average Max. Force $F_z$ [N]** | **Standard Deviation [N]** |
|---|---|---|
| 90 | 1.17 | 0.59 |
| 80 | 1.56 | 0.82 |
| 70 | 2.25 | 1.37 |
| 60 | 2.92 | 1.30 |
| 50 | 3.49 | 1.88 |
| 40 | 3.84 | 2.91 |

Table 3.1: Comparison of Pulling Force for Different Spine Angles

### 3.4.1 Comparing $F_z/F_x$ ratio to existing literature

The ratio $F_z/F_x$ compares the maximum pulling force to the tangential force applied to the rock's surface. Higher values of this ratio indicate a more efficient mechanism, as less force is required to push the microspines into the rock while achieving the same pulling force that the gripper can withstand. Therefore, a higher $F_z/F_x$ ratio is desirable for optimal performance.

| | **Pulling Force $F_z$** | **Tangential Force $F_x$** | **$F_z/F_x$ Ratio** |
|---|---|---|---|
| Gripper 1 (single microspine) | 20 N | 77 N | 0.26 |
| Gripper 2 (single microspine) | 3.5 N | 16 N | 0.22 |
| Our Gripper (microspine block) | 3.49 N | 5 N | 0.7 |

Table 3.2: Comparison of normal and tangential forces, and the $F_z/F_x$ ratio.

Gripper 1 [20] has a ratio of 0.26 and Gripper 2 [16] has a ratio of 0.22. The tests were only conducted with a single microspine. While our ratio of 0.7 is considerably higher, our test was conducted with the whole microspine block, which has nine spines implemented. From observations on average two to three microspines could attach to the rock per measurement. Correcting the ratio for a single spine, the new value would lay between 0.23 and 0.35 which is better than Gripper 2 and also possibly better than Gripper 1.
These results strongly depend on the type of rock tested, so it is important to acknowledge that variations in rock properties can significantly affect the performance.

Gripper 1 and Gripper 2 conducted tests only on porous volcanic rock, whereas we conducted tests on six different rock types. This variability in rock types could also affect the results.
(Better ratios could possibly be achieved by choosing harder spines, different spine geometries, and investigating the distances between the spines.)

## 3.5 Hook Mechanism Test

Not only the microspine block was tested, but also different hook mechanisms. The testing setup was similar to the one explained in section 3.3.

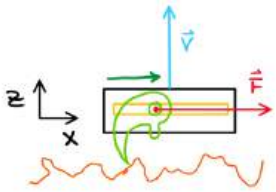


Figure 3.8: Only sliding Hook

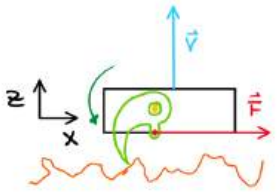


Figure 3.9: Only rotating Hook

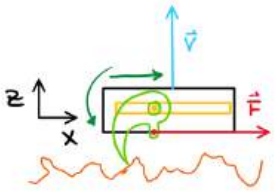


Figure 3.10: Combined sliding & rotating Hook

The green arrows in Figures 3.8, 3.9 and 3.10 indicate the degrees of freedom for the hooks to move. The first hook can only slide, the second hook can only rotate and the third can both slide and rotate. A spring is connected to the hooks and pulled with a force sensor (WH-A04 Portable Luggage Scale). With this sensor, a constant force of approximately $5N$ is applied to the hooks in the x-direction. Then the whole mechanism is pulled up by the Franka arm in z-direction with the same commanded velocity as in section 3.3. At the same time, the pulling force in z-direction is measured and logged to a file.

## 3.6 Results for Hook Mechanism Test

The tests for the individual hook mechanisms were tested three times per rock type. So per mechanism in total 18 measurements were logged and the maximum pulling force was determined.

| **Mechanism** | **Average Max. Force $F_z$ [N]** | **Standard Deviation [N]** |
|---|---|---|
| Sliding Hook | 3.11 | 1.72 |
| Rotating Hook | 5.15 | 2.77 |
| Combined Hook | 5.31 | 2.22 |

Table 3.3: Comparison of Maximum Pulling Force for Different Mechanisms

Even though the hook mechanism that can both slide and rotate has the highest average pulling force $F_z$ with $5.31N$, because of simplicity and to avoid complexity

the only rotating hook with a $F_z$ of $5.15N$ shows promising results.
Comparing the microspine block with a max. pulling force of $3.49N$ to the rotating hook with a max. pulling force of $5.15N$ reveals, that although the rotating hook mechanism shows a higher pulling force, the the microspine block has a higher chance of gripping because of the number of microspines.

## 3.7 Whole Gripper Test

For the final test, the whole gripper was evaluated. For this, the gripper was assembled and placed randomly on a rock. The gripper was assembled and placed on the rock. Although the placement was done manually, efforts were made to maintain randomness by varying the initial contact points and angles. The gripper was then deployed using the motor. After the gripper has established adhesion, it is then lifted using a force sensor in series as seen in Figure 3.11. Due to limitations in the strength of the 3D printed differential part, only 15% of the motor's stall torque was used, which would correspond to a torque of approximately $0.1\frac{N}{m}$.

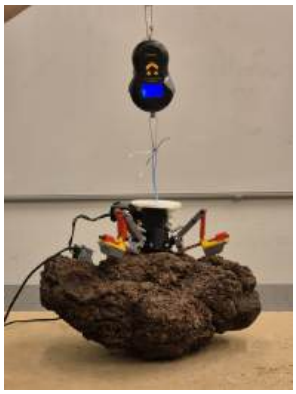

Figure 3.11: Whole Gripper Test Setup

Figure 3.12: Closeup of Gripper: Gripper attached to the Rock

When the used motor torque was increased to 25% the differential part failed.

Figure 3.13: Failed Differential Part

## 3.8 Result of whole Gripper Test

Because of time reasons, not too many tests could be conducted. The maximum pulling force $F_z$ measured was $14.42N$ and the average pulling force was $8.0N$ (without the gripper's self-weight of approximately $430g$. Comparing this result of a maximum pulling force of $14.42N$ to the tests from the spiny gripper [14], which managed a normal pulling force of up to $39.7N$ highlights a significant difference in

performance. This discrepancy suggests that the design and material properties of the spiny gripper allow for substantially higher normal pulling forces, which may be attributed to a bigger angle $\beta$ (the closing angle of the gripping legs) and better material strength due to optimized design. Additionally, the fact that only 15% of the maximum motor torque was utilized in our tests further indicates that there is considerable potential for improvement. By addressing material limitations and choosing higher motor torques, it is likely that the performance of our gripper could be significantly enhanced.

# Chapter 4

# Conclusion and Outlook

## 4.1 Conclusion

The scope of this bachelor thesis was to develop a robotic gripping mechanism, suitable for microgravity applications. The robot should further be integratable into robotic platforms such as SpaceHopper.
The functionality of the gripper, including an impact detection device, were to be tested at ETH as well as a microgravity set-up.
In the course of the project, the development of the gripper and testing at ETH were achieved, whereas the impact detection and testing in microgravity had to be scrapped due to the given time constraints. Figure 4.2 sums up which initial requirements were met and which had to be discarded.

With the maximum achieved gripping force of $14.42N$ per gripper, a total gripping force of $43N$ can be approximated, assuming that three full grippers make full contact on a rock surface. With this gripping force, SpaceHopper can hang upside down on a surface in any environment that has a gravitational acceleration smaller than $5ms^{-2}$.

## 4.2 Outlook / Future Work

While developing the gripper, a few limitations occurred. For example, the hardware for the final gripper design was not delivered in time, and the delivered springs were the wrong size. This compromised the design to be sub-optimal, since the 3D-printed parts introduce a higher friction and lower accuracy into the design than the ordered sliding shafts. On top of that, due to the faulty springs, the spines' compliance, a crucial design parameter, could not be implemented.

Due to time constraints, the final design could not be optimised. A next iteration of the gripper could integrate a macro hook mechanism as shown in section 3.5, since it shows promising attachment capabilities and great pulling forces.

Only 15% of the motor torque could be used, due to the limitations in the strength of the materials. Especially the 3D printed motor shaft and differential part constrained the use of higher motor torques. For future work, machined parts out of steel or other metals could seriously improve the gripping capabilities.
An additional issue that needs to be addressed is the seamless connection of the nylon tendon used in the differential to form a continuous loop. It is crucial that the tendon remains smooth and free of any irregularities or bumps to ensure optimal

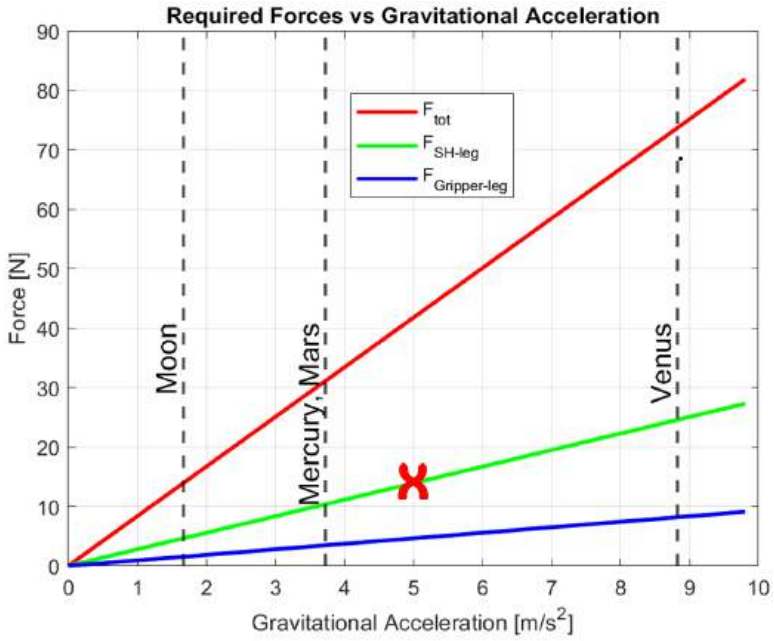


Figure 4.1: Required normal gripping force vs. gravitational acceleration $[ms^{-2}]$. The red cross indicates the max reached pulling force the gripper achieved.

| Initial project objectives | Achieved? |
|---|---|
| Gripping foot adjusting to surface | Yes |
| Gripping foot pulling parallel to surface | Yes |
| Simple design | Yes |
| Establish gripping force of 8.18N per gripper arm | No |
| Microgripping + Macrogripping | No |
| Automatic gripping initiation upon impact | No |
| Testing in microgravity | No |

Figure 4.2: Table of achievements that were met or not met.

performance.

Regarding the test of the whole gripper, additional measurements are needed to evaluate the reliability and performance of the gripper. It is essential to consider and measure various pull-off angles, especially investigating the pulling forces of the gripper in tangential direction w.r.t. the rock's surface. Furthermore, the number of tests conducted for each rock type should be increased to ensure more comprehensive and reliable results.

# Bibliography


[1] A. Parness, "Anchoring foot mechanisms for sampling and mobility in microgravity," in *2011 IEEE International Conference on Robotics and Automation*, May 2011, pp. 6596–6599, iSSN: 1050-4729.

[2] J. Brophy, R. Gershman, D. Landau, J. Polk, C. Porter, D. Yeomans, C. Allen, W. Williams, and E. Asphaug, *Asteroid Return Mission Feasibility Study*, Jul. 2011.

[3] E. Krasnova, "Sustainable operation and thermal concept for spacehopper," 2022, supervised by Prof. Dr. Marco Hutter, Dr. Hendrik Kolvenbach, Giorgio Valsecchi.

[4] A. Fujiwara, J. Kawaguchi, D. K. Yeomans, M. Abe, T. Mukai, T. Okada, J. Saito, H. Yano, M. Yoshikawa, D. J. Scheeres, O. Barnouin-Jha, A. F. Cheng, H. Demura, R. W. Gaskell, N. Hirata, H. Ikeda, T. Kominato, H. Miyamoto, A. M. Nakamura, R. Nakamura, S. Sasaki, and K. Uesugi, "The Rubble-Pile Asteroid Itokawa as Observed by Hayabusa," *Science*, vol. 312, no. 5778, pp. 1330–1334, Jun. 2006, publisher: American Association for the Advancement of Science.

[5] S. B. Backus, R. Onishi, A. Bocklund, A. Berg, E. D. Contreras, and A. Parness, "Design and testing of the JPL-Nautilus Gripper for deep-ocean geological sampling," *Journal of Field Robotics*, vol. 37, no. 6, pp. 972–986, 2020, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21934.

[6] "Stanford's New Spiny Grippers Will Help RoboSimian Go Rock Climbing - IEEE Spectrum."

[7] T. G. Chen, S. Newdick, J. Di, C. Bosio, N. Ongole, M. Lapôtre, M. Pavone, and M. R. Cutkosky, "Locomotion as manipulation with reachbot," *Science Robotics*, vol. 9, no. 89, p. eadi9762, 2024.

[8] P. Nadan, S. Backus, and A. M. Johnson, "Loris: A lightweight free-climbing robot for extreme terrain exploration," in *Proceedings of (ICRA) International Conference on Robotics and Automation*. IEEE, May 2024.

[9] M. J. Spenko, G. C. Haynes, J. A. Saunders, M. R. Cutkosky, A. A. Rizzi, R. J. Full, and D. E. Koditschek, "Biologically inspired climbing with a hexapedal robot," *Journal of Field Robotics*, vol. 25, no. 4-5, pp. 223–242, 2008.

[10] K. Uno, N. Takada, T. Okawara, K. Haji, A. Candalot, W. F. R. Ribeiro, K. Nagaoka, and K. Yoshida, "HubRobo: A Lightweight Multi-Limbed Climbing Robot for Exploration in Challenging Terrain," in *2020 IEEE-RAS 20th International Conference on Humanoid Robots (Humanoids)*, Jul. 2021, pp. 209–215, iSSN: 2164-0580.

[11] Y. Tanaka, Y. Shirai, A. Schperberg, X. Lin, and D. Hong, *SCALER: Versatile Multi-Limbed Robot for Free-Climbing in Extreme Terrains*, Dec. 2023.

[12] A. Parness, M. Frost, N. Thatte, J. King, K. Witkoe, M. Nevarez, M. Garrett, H. Aghazarian, and B. Kennedy, "Gravity-independent rock-climbing robot and a sample acquisition tool with microspine grippers," *Journal of Field Robotics*, vol. 30, 11 2013.

[13] A. Parness, N. Abcouwer, C. Fuller, N. Wiltsie, J. Nash, and B. Kennedy, "Lemur 3: A limbed climbing robot for extreme terrain mobility in space," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 5467–5473.

[14] P. Zi, K. Xu, Y. Tian, and X. Ding, "A mechanical adhesive gripper inspired by beetle claw for a rock climbing robot," *Mechanism and Machine Theory*, vol. 181, p. 105168, 2023.

[15] K. Nagaoka, H. Minote, K. Maruya, Y. Shirai, K. Yoshida, T. Hakamada, H. Sawada, and T. Kubota, "Passive Spine Gripper for Free-Climbing Robot in Extreme Terrain," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1765–1770, Jul. 2018, conference Name: IEEE Robotics and Automation Letters.

[16] A. Parness, A. Willig, A. Berg, M. Shekels, V. Arutyunov, C. Dandino, and B. Kennedy, "A microspine tool: Grabbing and anchoring to boulders on the Asteroid Redirect Mission," in *2017 IEEE Aerospace Conference*, Mar. 2017, pp. 1–10.

[17] A. T. Asbeck, S. Kim, M. R. Cutkosky, W. R. Provancher, and M. Lanzetta, "Scaling hard vertical surfaces with compliant microspine arrays," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1165–1179, 2006.

[18] A. T. Asbeck and M. R. Cutkosky, "Designing Compliant Spine Mechanisms for Climbing," *Journal of Mechanisms and Robotics*, vol. 4, no. 3, p. 031007, Aug. 2012.

[19] Franka Emika GmbH, "Panda - Datasheet," https://www.wiredworkers.io/wp-content/uploads/2019/12/Panda_FrankaEmika_ENG.pdf, 2019, [Online; accessed June 24, 2024].

[20] Q. Cheng, W. Zhang, W. Zhou, J. Li, Z. Li, T. Yu, B. Wang, and S. Wang, "Design and analysis of a bionic adhesion coring sampler for space unstructured surface," *Acta Astronautica*, vol. 215, pp. 79–101, Feb. 2024.

[21] A. Parness, M. Frost, N. Thatte, J. P. King, K. Witkoe, M. Nevarez, M. Garrett, H. Aghazarian, and B. Kennedy, "Gravity-independent Rock-climbing Robot and a Sample Acquisition Tool with Microspine Grippers," *Journal of Field Robotics*, vol. 30, no. 6, pp. 897–915, 2013, _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21476.

[22] Engineering ToolBox, "Toggle joint," https://www.engineeringtoolbox.com/toggle-joint-d_2077.html, accessed: 24-Jun-2024.

# Appendix A

# Calculations

## A.1 MATLAB Force Requirement

```
m_sh = 7 % [kg]
m_gripper = 0.450 % [kg]
m_tot = m_sh+3*m_gripper

% Define the range for gravitational acceleration
g = linspace(0, 9.81, 100);


% Forces
F_tot = m_tot .* g;
F_leg = F_tot / 3;
F_arm = F_leg / 3;

% Gravitational parameters to mark [m/s^2]
g_parameters = [0.38, 0.9, 0.17] * 9.81;
g_labels = {'Mercury, Mars', 'Venus', 'Moon'};

% Plot Force
figure;
plot(g, F_tot, 'r', 'LineWidth', 2); hold on;
plot(g, F_leg, 'g', 'LineWidth', 2);
plot(g, F_arm, 'b', 'LineWidth', 2);


for i = 1:length(g_parameters)
    xline(g_parameters(i), '--k', 'LineWidth', 1.5);
end


xlabel('Gravitational Acceleration [m/s^2]');
ylabel('Force [N]');
title('Required Forces vs Gravitational Acceleration');
legend('F_{tot}', 'F_{SH-leg}', 'F_{Gripper-leg}');
grid on;

for i = 1:length(g_parameters)
```

```
    text(g_parameters(i), max(F_tot)/2, g_labels{i}, ...
        'Rotation', 90, 'VerticalAlignment', 'bottom', 'HorizontalAlignment', 'right', 'Fo
end

hold off;
```

```
m_sh =

     7


m_gripper =

    0.4500


m_tot =

    8.3500
```

## A.2 Saint-Venant's Principle

$\frac{L}{D_i} > 1$, 1.6 : 1 good, 3 : 1 very good.

When designing a slider, this ratio should be kept such that the sliding mechanism does not jam. For this project, as a compromise of sliding capability and size, a ratio of 2 : 1 was chosen.

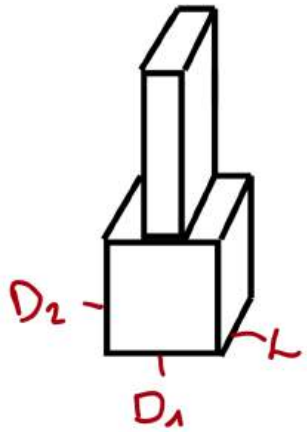


Figure A.1: Dimensions of a slider insert. To not jam, $D_i > 1.6L$, according to the principle of Saint-Venant

## A.3 Dimensioning of Links

Below, the calculations used for dimensioning the links of the mechanism can be seen. It is important to notice, that the simplification was made that the force

pushing down on the mechanism is assumed to be perfectly vertical. In reality, it acts perpendicular to the link transmitting the force. The tangential forces P were calculated using the model of a toggle-joint [22].

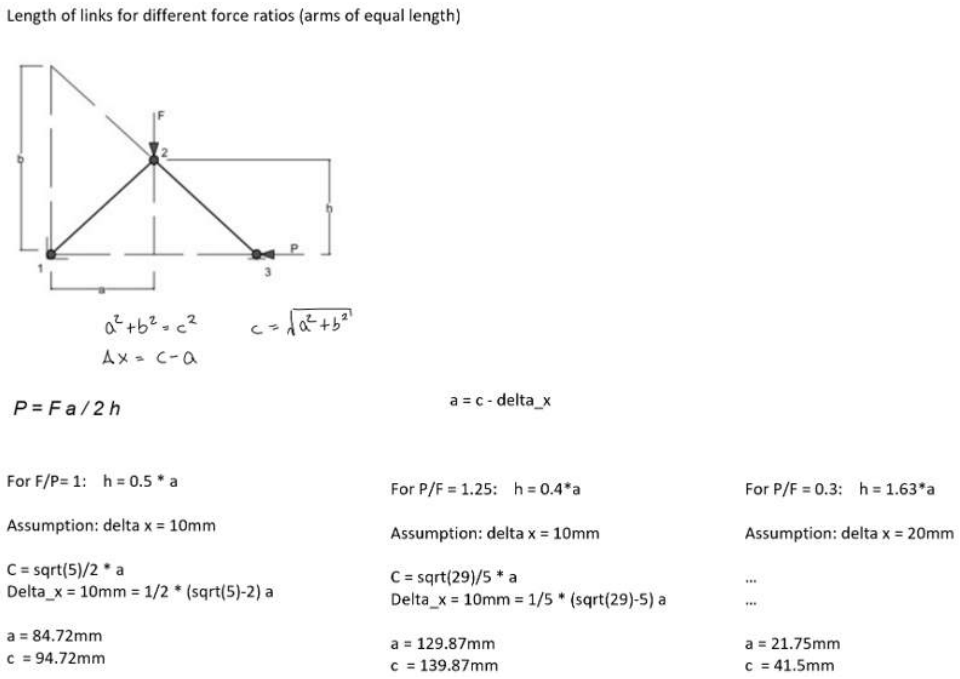


Figure A.2: Calculations for the length of links, with the assumption of force ratio and delta x.

## A.4 Arduino Code for Dynamixel AX-12A Control

```
#include <DynamixelShield.h>

/*
Baud Rate Values for Dynamixel Motors:

Value   Baud Rate (bps)    Margin of Error
1       1M                  0.000%
3       500,000             0.000%
4       400,000             0.000%
7       250,000             0.000%
9       200,000             0.000%
16      115200              -2.124%
34      57600               0.794%
103     19200               -0.160%
207     9600                -0.160%
*/

#if defined(ARDUINO_AVR_UNO) || defined(ARDUINO_AVR_MEGA2560)
  #include <SoftwareSerial.h>
  SoftwareSerial soft_serial(7, 8); // DYNAMIXELShield UART RX/TX
  #define DEBUG_SERIAL soft_serial
#elif defined(ARDUINO_SAM_DUE) || defined(ARDUINO_SAM_ZERO)
  #define DEBUG_SERIAL SerialUSB
#else
  #define DEBUG_SERIAL Serial
```

```
26 #endif
27
28 //This example is written for DYNAMIXEL AX-12A with Protocol 1.0.
29
30 #define OPERATING_MODE_ADDR_LEN     2
31
32 /*The Control Table is divided into 2 Areas.
33 Data in the RAM Area is reset to initial values
34 when the power is reset(Volatile). On the other
35 hand, data in the EEPROM Area is maintained even
36 when the device is powered off(Non-Volatile).*/
37
38 //Control Table of EEPROM Area
39 #define MODEL_NUMBER_ADDR           0
40 #define MODEL_NUMBER_ADDR_LEN       2
41 #define FIRMWARE_VERSION_ADDR       2
42 #define FIRMWARE_VERSION_ADDR_LEN   1
43 #define ID_ADDR                     3
44 #define ID_ADDR_LEN                 1
45 #define BAUDRATE_ADDR               4
46 #define BAUDRATE_ADDR_LEN           1
47 #define RETURN_DELAY_TIME_ADDR      5
48 #define RETURN_DELAY_TIME_ADDR_LEN  1
49 #define CW_ANGLE_LIMIT_ADDR         6
50 #define CW_ANGLE_LIMIT_ADDR_LEN     2
51 #define CCW_ANGLE_LIMIT_ADDR        8
52 #define CCW_ANGLE_LIMIT_ADDR_LEN    2
53 #define TEMPERATURE_LIMIT_ADDR      11
54 #define TEMPERATURE_LIMIT_ADDR_LEN  1
55 #define MIN_VOLTAGE_LIMIT_ADDR      12
56 #define MIN_VOLTAGE_LIMIT_ADDR_LEN  1
57 #define MAX_VOLTAGE_LIMIT_ADDR      13
58 #define MAX_VOLTAGE_LIMIT_ADDR_LEN  1
59 #define MAX_TORQUE_ADDR             14
60 #define MAX_TORQUE_ADDR_LEN         2
61 #define STATUS_RETURN_LEVEL_ADDR    16
62 #define STATUS_RETURN_LEVEL_ADDR_LEN 1
63 #define ALARM_LED_ADDR              17
64 #define ALARM_LED_ADDR_LEN          1
65 #define SHUTDOWN_ADDR               18
66 #define SHUTDOWN_ADDR_LEN           1
67
68 //Control Table of RAM Area
69 #define TORQUE_ENABLE_ADDR           24
70 #define TORQUE_ENABLE_ADDR_LEN       1
71 #define LED_ADDR                     25
72 #define LED_ADDR_LEN                 1
73 #define CW_COMPLIANCE_MARGIN_ADDR    26
74 #define CW_COMPLIANCE_MARGIN_ADDR_LEN 1
75 #define CCW_COMPLIANCE_MARGIN_ADDR   27
76 #define CCW_COMPLIANCE_MARGIN_ADDR_LEN 1
77 #define CW_COMPLIANCE_SLOPE_ADDR     28
78 #define CW_COMPLIANCE_SLOPE_ADDR_LEN 1
79 #define CCW_COMPLIANCE_SLOPE_ADDR    29
80 #define CCW_COMPLIANCE_SLOPE_ADDR_LEN 1
81 #define GOAL_POSITION_ADDR           30
82 #define GOAL_POSITION_ADDR_LEN       2
83 #define MOVING_SPEED_ADDR            32
84 #define MOVING_SPEED_ADDR_LEN        2
85 #define TORQUE_LIMIT_ADDR            34
86 #define TORQUE_LIMIT_ADDR_LEN        2
87 #define PRESENT_POSITION_ADDR        36
88 #define PRESENT_POSITION_ADDR_LEN    2
89 #define PRESENT_SPEED_ADDR           38
90 #define PRESENT_SPEED_ADDR_LEN       2
91 #define PRESENT_LOAD_ADDR            40
92 #define PRESENT_LOAD_ADDR_LEN        2
```

```
93  #define PRESENT_VOLTAGE_ADDR          42
94  #define PRESENT_VOLTAGE_ADDR_LEN      1
95  #define PRESENT_TEMPERATURE_ADDR      43
96  #define PRESENT_TEMPERATURE_ADDR_LEN  1
97  #define REGISTERED_ADDR               44
98  #define REGISTERED_ADDR_LEN           1
99  #define MOVING_ADDR                   46
100 #define MOVING_ADDR_LEN               1
101 #define LOCK_ADDR                     47
102 #define LOCK_ADDR_LEN                 1
103 #define PUNCH_ADDR                    48
104 #define PUNCH_ADDR_LEN                2
105
106 #define TIMEOUT 100    //default communication timeout 10ms
107
108
109
110 const uint8_t DXL_ID = 1;
111 const float DXL_PROTOCOL_VERSION = 1.0;
112 const uint32_t baudrate = 1000000;
113
114 DynamixelShield dxl;
115
116 //use as parameters for some configurations
117
118 uint8_t returned_id = 0;
119 uint8_t returned_baudrate = 0;
120 uint16_t present_position = 0;
121 uint16_t present_speed = 0;
122
123 uint8_t turn_on = 1;
124 uint8_t turn_off = 0;
125
126 uint16_t goalPosition1 = 300;
127 uint16_t goalPosition2 = 500;
128
129 float Kp = 2; // Proportional gain
130 uint16_t goal_position = 1022; // Default goal position (middle of the
      range)
131
132
133 void setup() {
134   DEBUG_SERIAL.begin(115200);   // Set debugging port baudrate to
        115200bps
135   while(!DEBUG_SERIAL);         // Wait until the serial port for
        terminal is opened
136
137   // Set Port baudrate to 1000000bps. This has to match with DYNAMIXEL
        baudrate (AX-12A).
138   dxl.begin(baudrate);
139
140   // Set Port Protocol Version. This has to match with DYNAMIXEL
        protocol version.
141   dxl.setPortProtocolVersion(DXL_PROTOCOL_VERSION);
142
143   DEBUG_SERIAL.print("Read for PROTOCOL ");
144   DEBUG_SERIAL.print(DXL_PROTOCOL_VERSION, 1);
145   DEBUG_SERIAL.print(", ID ");
146   DEBUG_SERIAL.println(DXL_ID);
147
148   // Read DYNAMIXEL ID
149   dxl.read(DXL_ID, ID_ADDR, ID_ADDR_LEN, (uint8_t*)&returned_id,
        sizeof(returned_id), TIMEOUT);
150   DEBUG_SERIAL.print("ID : ");
151   DEBUG_SERIAL.println(returned_id);
152   delay(100);
153
```

```
154   // Read DYNAMIXEL Baudrate
155   dxl.read(DXL_ID, BAUDRATE_ADDR, BAUDRATE_ADDR_LEN, (uint8_t*)&
          returned_baudrate, sizeof(returned_baudrate), TIMEOUT);
156   DEBUG_SERIAL.print("Baud Rate : ");
157   DEBUG_SERIAL.println(returned_baudrate);
158   delay(100);
159
160   // Read DYNAMIXEL Present Position
161   dxl.read(DXL_ID, PRESENT_POSITION_ADDR, PRESENT_POSITION_ADDR_LEN, (
          uint8_t*)&present_position, sizeof(present_position), TIMEOUT);
162   DEBUG_SERIAL.print("Present Position : ");
163   DEBUG_SERIAL.println(present_position);
164
165   // It configures the Dynamixel motor to joint mode, limiting its
          rotation within specified angles.
166   delay(10);
167
168   // Turn off torque when configuring items in EEPROM area
169   if(dxl.write(DXL_ID, TORQUE_ENABLE_ADDR, (uint8_t*)&turn_off ,
          TORQUE_ENABLE_ADDR_LEN, TIMEOUT))
170     DEBUG_SERIAL.println("DYNAMIXEL Torque off");
171   else
172     DEBUG_SERIAL.println("Error: Torque off failed");
173
174   delay(10);
175
176   // Set CW and CCW angle limits for joint mode (min position = 300,
          max position = 700)
177   uint16_t min_position = 0;
178   uint16_t max_position = 1023; //0; //
179   if(dxl.write(DXL_ID, CW_ANGLE_LIMIT_ADDR, (uint8_t*)&min_position,
          CW_ANGLE_LIMIT_ADDR_LEN, TIMEOUT)
180       && dxl.write(DXL_ID, CCW_ANGLE_LIMIT_ADDR, (uint8_t*)&
              max_position, CCW_ANGLE_LIMIT_ADDR_LEN, TIMEOUT))
181     DEBUG_SERIAL.println("Set operating mode");
182   else
183     DEBUG_SERIAL.println("Error: Set operating mode failed");
184
185   delay(10);
186
187   // Set goal position to 300
188   uint16_t goal_position = 1023;
189   if(dxl.write(DXL_ID, GOAL_POSITION_ADDR, (uint8_t*)&goal_position,
          GOAL_POSITION_ADDR_LEN, TIMEOUT))
190     DEBUG_SERIAL.println("Set goal position");
191   else
192     DEBUG_SERIAL.println("Error: Set goal position failed / Or motor
          is in wheel mode!");
193
194   delay(10);
195
196
197
198   // Set maximum torque
199   uint16_t max_torque = 1023 * 0.15; // 15% of max torque (1023)
200   if(dxl.write(DXL_ID, MAX_TORQUE_ADDR, (uint8_t*)&max_torque,
          MAX_TORQUE_ADDR_LEN, TIMEOUT))
201     DEBUG_SERIAL.println("Set max torque");
202   else
203     DEBUG_SERIAL.println("Error: Set max torque failed");
204
205   delay(10);
206
207   // Set torque limit
208   uint16_t torque_limit = 1023 * 0.15; // 15% of max torque (1023)
209   if(dxl.write(DXL_ID, TORQUE_LIMIT_ADDR, (uint8_t*)&torque_limit,
          TORQUE_LIMIT_ADDR_LEN, TIMEOUT))
```

```
210       DEBUG_SERIAL.println("Set torque limit");
211     else
212       DEBUG_SERIAL.println("Error: Set torque limit failed");
213
214     delay(10);
215
216     // Turn on torque
217     if(dxl.write(DXL_ID, TORQUE_ENABLE_ADDR, (uint8_t*)&turn_on,
            TORQUE_ENABLE_ADDR_LEN, TIMEOUT))
218       DEBUG_SERIAL.println("Torque on");
219     else
220       DEBUG_SERIAL.println("Error: Torque on failed");
221   }
222
223
224   void loop() {
225
226     delay(20);
227     //readLoad();
228     //controlMotor();
229     //readSpeed();
230     controlMotorInJointMode();
231     //currentPosition();
232     //controlLoop();
233     //updateGoalPosition();
234
235   }
236
237   /*
238   void moveMotor(uint16_t start, uint16_t end) {
239     uint16_t currentPos = start;
240     while (currentPos != end) {
241       //DEBUG_SERIAL.print("Moving to position: ");
242       //DEBUG_SERIAL.println(end);
243
244       dxl.write(DXL_ID, GOAL_POSITION_ADDR, (uint8_t*)&end,
              GOAL_POSITION_ADDR_LEN, TIMEOUT);
245
246       // Continuously read present position until goal is reached
247       do {
248         dxl.read(DXL_ID, PRESENT_POSITION_ADDR,
                PRESENT_POSITION_ADDR_LEN, (uint8_t*)&currentPos, sizeof(
                currentPos), TIMEOUT);
249         //DEBUG_SERIAL.print("Current Position: ");
250         DEBUG_SERIAL.println(currentPos);
251         delay(100); // Adjust delay according to your motor's speed
252       } while (currentPos != end);
253     }
254   }
255   */
256
257
258   void readLoad() {
259     uint16_t loadValue = 0;
260     dxl.read(DXL_ID, PRESENT_LOAD_ADDR, PRESENT_LOAD_ADDR_LEN, (uint8_t
            *)&loadValue, sizeof(loadValue), TIMEOUT);
261     //DEBUG_SERIAL.print("Present Load: ");
262     //DEBUG_SERIAL.println(loadValue);
263     float load = convertLoadToPercentage(loadValue);
264     DEBUG_SERIAL.println(loadValue);
265     //DEBUG_SERIAL.println(load);
266   }
267
268   float convertLoadToPercentage(int loadValue) {
269     // Convert load value to percentage
270     float percentage = 0.0;
271
```

```
272   // Check direction
273   if (loadValue < 1024) {
274     // CCW direction
275     percentage = (loadValue / 10.23); // Convert to percentage
276   } else {
277     // CW direction
278     percentage = ((1024 - loadValue) / 10.23); // Convert to
          percentage
279   }
280
281   return percentage;
282 }
283
284
285 uint16_t calculateSpeed(uint8_t percentage, bool clockwise) {
286   if (percentage > 100) percentage = 100;
287   uint16_t speed = (percentage * 1023) / 100; // Scale 0-100 to 0-1023
288   if (clockwise) {
289     speed += 1024; // Offset for clockwise direction
290   }
291   return speed;
292 }
293
294
295 /*
296 void readSpeed() {
297   dxl.read(DXL_ID, PRESENT_SPEED_ADDR, PRESENT_SPEED_ADDR_LEN, (
        uint8_t*)&present_speed, sizeof(present_speed), TIMEOUT);
298
299   int speedValue = present_speed & 0x3FF; // Mask out the direction
        bit (10th bit)
300   bool clockwise = (present_speed & 0x400) != 0; // Check the
        direction bit
301
302   int speedPercentage = (speedValue * 1000) / 1023; // Convert to
        percentage
303
304   if (clockwise) {
305     DEBUG_SERIAL.print("Current Speed: ");
306     DEBUG_SERIAL.print(speedPercentage);
307     DEBUG_SERIAL.println(" (CW)");
308   } else {
309     speedPercentage = -speedPercentage; // Make it negative for CCW
310     DEBUG_SERIAL.print("Current Speed: ");
311     DEBUG_SERIAL.print(speedPercentage);
312     DEBUG_SERIAL.println(" (CCW)");
313   }
314 }
315 */
316
317 /*
318 void controlMotor() {
319   static bool upPressed = false;
320   static bool downPressed = false;
321   static uint8_t speedPercentage = 5; // Default speed percentage
        (100% of max speed)
322
323   if (Serial.available() > 0) {
324     char command = Serial.read();
325     if (command == 'u') {
326       upPressed = true;
327       downPressed = false;
328     } else if (command == 'd') {
329       downPressed = true;
330       upPressed = false;
331     } else if (command == 's') { // Stop command
332       upPressed = false;
```

```
333         downPressed = false;
334       } else if (command >= '0' && command <= '9') {
335         speedPercentage = (command - '0'); // * 10; // Set speed
                percentage (0-90%)
336       }
337     }
338
339     if (upPressed) {
340       uint16_t speed = calculateSpeed(speedPercentage, true);
341       dxl.write(DXL_ID, MOVING_SPEED_ADDR, (uint8_t*)&speed,
              MOVING_SPEED_ADDR_LEN, TIMEOUT);
342     } else if (downPressed) {
343       uint16_t speed = calculateSpeed(speedPercentage, false);
344       dxl.write(DXL_ID, MOVING_SPEED_ADDR, (uint8_t*)&speed,
              MOVING_SPEED_ADDR_LEN, TIMEOUT);
345     } else {
346       uint16_t speed = 0; // Stop the motor
347       dxl.write(DXL_ID, MOVING_SPEED_ADDR, (uint8_t*)&speed,
              MOVING_SPEED_ADDR_LEN, TIMEOUT);
348     }
349   }
350   */
351
352   void controlMotorInJointMode() {
353     static uint16_t goal_position = 1022; // Default goal position (
          middle of the range)
354     static uint8_t speedPercentage = 100; // Default speed percentage
          (100% of max speed)
355     bool positionChanged = false;
356     uint16_t current_position = readPosition();
357
358     if (Serial.available() > 0) {
359       char command = Serial.read();
360       if (command == 'u' && goal_position < 1013) {
361         goal_position += 10; // Increase goal position
362         positionChanged = true;
363       } else if (command == 'd' && goal_position > 10) {
364         goal_position -= 10; // Decrease goal position
365         positionChanged = true;
366       } else if (command >= '0' && command <= '9') {
367         speedPercentage = (command - '0') * 10; // Set speed percentage
              (0-90%)
368       }
369     }
370
371     if (positionChanged) {
372       dxl.write(DXL_ID, GOAL_POSITION_ADDR, (uint8_t*)&goal_position,
              GOAL_POSITION_ADDR_LEN, TIMEOUT);
373       uint16_t speed = calculateSpeed(speedPercentage, true);
374       dxl.write(DXL_ID, MOVING_SPEED_ADDR, (uint8_t*)&speed,
              MOVING_SPEED_ADDR_LEN, TIMEOUT);
375       //DEBUG_SERIAL.print("Goal Position: ");
376       //DEBUG_SERIAL.println(goal_position);
377       //Serial.print("   Present Position: ");
378       //Serial.println(current_position);
379     }
380   }
381
382
383   /*
384   void controlLoop() {
385     // Read current position
386     uint16_t current_position = readPosition();
387
388     // Calculate error
389     int16_t error = goal_position - current_position;
390
```

```
391   // Calculate speed using proportional control
392   int16_t speed = Kp * error;
393
394   // Set the speed to drive the motor
395   //dxl.write(DXL_ID, MOVING_SPEED_ADDR, (uint8_t*)&speed,
          MOVING_SPEED_ADDR_LEN, TIMEOUT);
396   // Set the speed and direction to drive the motor
397   if (speed >= 0) {
398     dxl.write(DXL_ID, MOVING_SPEED_ADDR, (uint8_t*)&speed,
            MOVING_SPEED_ADDR_LEN, TIMEOUT);
399   } else {
400     // Set direction bit for CW direction
401     speed = -speed; // Make speed positive
402     speed |= 0x400; // Set direction bit
403     dxl.write(DXL_ID, MOVING_SPEED_ADDR, (uint8_t*)&speed,
            MOVING_SPEED_ADDR_LEN, TIMEOUT);
404   }
405
406   // Print goal position and present position
407   Serial.print("Goal Position: ");
408   Serial.print(goal_position);
409   Serial.print("   Present Position: ");
410   Serial.println(current_position);
411
412   // Delay for stability
413   delay(100);
414 }
415 */
416
417 uint16_t readPosition() {
418   uint16_t current_position = 0;
419   dxl.read(DXL_ID, PRESENT_POSITION_ADDR, PRESENT_POSITION_ADDR_LEN, (
          uint8_t*)&current_position, sizeof(current_position), TIMEOUT);
420   return current_position;
421 }
422
423 void currentPosition() {
424   uint16_t current_position = 0;
425   dxl.read(DXL_ID, PRESENT_POSITION_ADDR, PRESENT_POSITION_ADDR_LEN, (
          uint8_t*)&current_position, sizeof(current_position), TIMEOUT);
426   //Serial.print("   Current Position: ");
427   //Serial.println(current_position);
428 }
429
430 void updateGoalPosition() {
431   if (Serial.available() > 0) {
432     char command = Serial.read();
433     if (command == 'u' && goal_position < 1013) {
434       goal_position += 10; // Increase goal position
435     } else if (command == 'd' && goal_position > 10) {
436       goal_position -= 10; // Decrease goal position
437     }
438   }
439 }
```

Listing A.1: Arduino code for Dynamixel AX-12A

## A.5 Python Code of Controller for Franka arm

### A.5.1 [y, z] Velocity and [x] Force Control Script

```
1 import rospy
2 from franka_example_controllers.msg import PoseWrenchCommand
3 from franka_msgs.msg import FrankaState
4 from std_srvs.srv import SetBool
5 import numpy as np
```

```
6
7
8   def get_force_command(force_x, force_y, force_z, quat_x, quat_y,
        quat_z, quat_w):
9       cmd = PoseWrenchCommand()
10      cmd.pose.orientation.x = quat_x
11      cmd.pose.orientation.y = quat_y
12      cmd.pose.orientation.z = quat_z
13      cmd.pose.orientation.w = quat_w
14      cmd.wrench.force.x = force_x
15      cmd.wrench.force.y = force_y
16      cmd.wrench.force.z = force_z
17      cmd.force_selection = [True, True, True]
18      return cmd
19
20
21  def get_tracking_command(pos_x, pos_y, pos_z, quat_x, quat_y, quat_z,
        quat_w):
22      cmd = PoseWrenchCommand()
23      cmd.pose.orientation.x = quat_x
24      cmd.pose.orientation.y = quat_y
25      cmd.pose.orientation.z = quat_z
26      cmd.pose.orientation.w = quat_w
27      cmd.pose.position.x = pos_x
28      cmd.pose.position.y = pos_y
29      cmd.pose.position.z = pos_z
30      cmd.force_selection = [False, False, False]
31      return cmd
32
33
34  def get_hybrid_command(force_x, pos_y, pos_z, quat_x, quat_y, quat_z,
        quat_w):
35      cmd = PoseWrenchCommand()
36      cmd.pose.orientation.x = quat_x
37      cmd.pose.orientation.y = quat_y
38      cmd.pose.orientation.z = quat_z
39      cmd.pose.orientation.w = quat_w
40      cmd.wrench.force.x = force_x
41      cmd.pose.position.y = pos_y
42      cmd.pose.position.z = pos_z
43      cmd.force_selection = [True, False, False]
44      return cmd
45
46
47  class RosControllerNode:
48      """Simple controller node for the dynaarm."""
49
50      def post_init(self):
51          default_orientation = [1, 0, 0, 0]
52
53          distance_y = abs(self.end_pos_y - self.start_pos_y)
54          distance_z = abs(self.end_pos_z - self.start_pos_z)
55          distance = np.sqrt(distance_y**2 + distance_z**2)
56          dt = self.timesteps
57          total_time = distance / self.desired_speed
58          num_steps = int(total_time / dt)
59
60          self.actions = {}
61          for i in range(num_steps):
62              t = (i + 1) * dt
63              pos_y = self.start_pos_y + (self.end_pos_y - self.
                    start_pos_y) * (t / total_time)
64              pos_z = self.start_pos_z + (self.end_pos_z - self.
                    start_pos_z) * (t / total_time)
65              self.actions[t] = get_hybrid_command(self.constant_force_x
                    , pos_y, pos_z, *default_orientation)
66
```

```
67          self.switch_times = list(self.actions.keys())
68          self.actions_msgs = list(self.actions.values())
69          self.current_action_idx = 0
70          self.logger_srv(True)
71          self.is_initialized = True
72
73
74      def _init_(self, dt) -> None:
75          self.timesteps = dt
76          self.elapsed_time = 0
77
78          self.is_initialized = False
79          self.cmd_topic = "/cartesian_impedance_example_controller/
                pose_wrench_command"
80          self.pub = rospy.Publisher(self.cmd_topic, PoseWrenchCommand)
81          # wait for service
82          rospy.wait_for_service("/measurement_logger/start_logging")
83          self.logger_srv = rospy.ServiceProxy("/measurement_logger/
                start_logging", SetBool)
84
85          self.current_position = None
86          self.state_sub = rospy.Subscriber("/franka_state_controller/
                franka_states", FrankaState, self.state_cb, queue_size=1)
87
88          self.delta_pos_y = 0.0
89          self.delta_pos_z = 0.3
90          self.start_pos_x = None
91          self.start_pos_y = None
92          self.start_pos_z = None
93
94          self.desired_speed = 0.001  # Desired absolute speed
95          self.constant_force_x = 5.0  # Constant force in x-direction
96          self.max_deviation_x = 0.1  # Maximum allowed deviation in x-
                direction
97          self.deviation_x = 0.0  # Initialize deviation_x
98
99
100     #get the current position and store it in a vector
101     def state_cb(self, msg):
102         pose = np.array(msg.O_T_EE, order="C").reshape(4,4).T
103         self.current_position = pose[:-1, -1]
104
105         # Update start positions if they are not initialized
106         if self.start_pos_x is None:
107             self.start_pos_x = self.current_position[0]  # x-
                    coordinate
108         if self.start_pos_y is None:
109             self.start_pos_y = self.current_position[1]  # y-
                    coordinate
110         if self.start_pos_z is None:
111             self.start_pos_z = self.current_position[2]  # z-
                    coordinate
112
113         # Calculate end positions relative to start positions
114         if self.start_pos_x is not None and self.start_pos_y is not
                None and self.start_pos_z is not None:
115             self.end_pos_y = self.start_pos_y + self.delta_pos_y
116             self.end_pos_z = self.start_pos_z + self.delta_pos_z
117
118     def process(self) -> None:
119         if self.start_pos_x is None: # No initial reading found
120             return
121         else:
122             if not self.is_initialized:
123                 self.post_init()
124
125         self.elapsed_time += self.timesteps
```

```
126         if self.elapsed_time >= self.switch_times[self.
                current_action_idx]:
127             self.current_action_idx += 1
128 
129             if self.current_action_idx >= len(self.switch_times):
130                 self.logger_srv(False)
131                 print("Done!")
132                 exit()
133 
134             print("Switching to action #", self.current_action_idx)
135         msg = self.actions_msgs[self.current_action_idx]
136 
137         default_orientation = [1, 0, 0, 0]
138         # Check deviation in x-direction from the starting position
139         if self.start_pos_x is None:
140             self.start_pos_x = self.current_position[0]  # Assuming x-
                coordinate is the first element
141         else:
142             self.deviation_x = abs(self.current_position[0] - self.
                start_pos_x)
143             if self.deviation_x > self.max_deviation_x:
144                 rospy.logerr("Exceeded maximum deviation in x-
                    direction. Stopping motion.")
145 
146                 # Set force in x direction to 0
147                 msg.wrench.force.x = 0.0
148                 exit()
149 
150         self.pub.publish(msg)
151 
152 
153 if _name_ == "_main_":
154     rospy.init_node("controller_node", anonymous=True)
155     frequency = rospy.get_param("~frequency", 30)
156     node = RosControllerNode(dt=1 / frequency)
157     rospy.loginfo(f"Controller node up and running. Running inference
            at {frequency} Hz.")
158     rospy.Timer(rospy.Duration(1.0 / frequency), lambda _: node.
            process())
159     rospy.spin()
```

Listing A.2: Controller Node Script 1

### A.5.2 [x, y, z] Velocity Control Script

```
1  import rospy
2  from franka_example_controllers.msg import PoseWrenchCommand
3  from franka_msgs.msg import FrankaState
4  from std_srvs.srv import SetBool
5  import numpy as np
6 
7 
8  def get_force_command(force_x, force_y, force_z, quat_x, quat_y,
       quat_z, quat_w):
9      cmd = PoseWrenchCommand()
10     cmd.pose.orientation.x = quat_x
11     cmd.pose.orientation.y = quat_y
12     cmd.pose.orientation.z = quat_z
13     cmd.pose.orientation.w = quat_w
14     cmd.wrench.force.x = force_x
15     cmd.wrench.force.y = force_y
16     cmd.wrench.force.z = force_z
17     cmd.force_selection = [True, True, True]
18     return cmd
19 
20 
21 def get_tracking_command(pos_x, pos_y, pos_z, quat_x, quat_y, quat_z,
       quat_w):
```

```
22      cmd = PoseWrenchCommand()
23      cmd.pose.orientation.x = quat_x
24      cmd.pose.orientation.y = quat_y
25      cmd.pose.orientation.z = quat_z
26      cmd.pose.orientation.w = quat_w
27      cmd.pose.position.x = pos_x
28      cmd.pose.position.y = pos_y
29      cmd.pose.position.z = pos_z
30      cmd.force_selection = [False, False, False]
31      return cmd
32
33
34  def get_hybrid_command(force_x, pos_y, pos_z, quat_x, quat_y, quat_z,
        quat_w):
35      cmd = PoseWrenchCommand()
36      cmd.pose.orientation.x = quat_x
37      cmd.pose.orientation.y = quat_y
38      cmd.pose.orientation.z = quat_z
39      cmd.pose.orientation.w = quat_w
40      cmd.wrench.force.x = force_x
41      cmd.pose.position.y = pos_y
42      cmd.pose.position.z = pos_z
43      cmd.force_selection = [True, False, False]
44      return cmd
45
46
47  class RosControllerNode:
48      """Simple controller node for the dynaarm."""
49
50      def post_init(self):
51          default_orientation = [1, 0, 0, 0]
52
53          distance_x = abs(self.end_pos_x - self.start_pos_x)
54          distance_y = abs(self.end_pos_y - self.start_pos_y)
55          distance_z = abs(self.end_pos_z - self.start_pos_z)
56          distance = np.sqrt(distance_x**2 + distance_y**2 + distance_z
                **2)
57          dt = self.timesteps
58          total_time = distance / self.desired_speed
59          num_steps = int(total_time / dt)
60
61          self.actions = {}
62          for i in range(num_steps):
63              t = (i + 1) * dt
64              pos_x = self.start_pos_x + (self.end_pos_x - self.
                    start_pos_x) * (t / total_time)
65              pos_y = self.start_pos_y + (self.end_pos_y - self.
                    start_pos_y) * (t / total_time)
66              pos_z = self.start_pos_z + (self.end_pos_z - self.
                    start_pos_z) * (t / total_time)
67              self.actions[t] = get_tracking_command(pos_x, pos_y, pos_z
                    , *default_orientation)
68
69          self.switch_times = list(self.actions.keys())
70          self.actions_msgs = list(self.actions.values())
71          self.current_action_idx = 0
72          self.logger_srv(True)
73          self.is_initialized = True
74
75
76      def _init_(self, dt) -> None:
77          self.timesteps = dt
78          self.elapsed_time = 0
79
80          self.is_initialized = False
81          self.cmd_topic = "/cartesian_impedance_example_controller/
                pose_wrench_command"
```

```
82          self.pub = rospy.Publisher(self.cmd_topic, PoseWrenchCommand)
83          # wait for service
84          rospy.wait_for_service("/measurement_logger/start_logging")
85          self.logger_srv = rospy.ServiceProxy("/measurement_logger/
                start_logging", SetBool)
86
87          self.current_position = None
88          self.state_sub = rospy.Subscriber("/franka_state_controller/
                franka_states", FrankaState, self.state_cb, queue_size=1)
89
90          self.delta_pos_x = 0
91          self.delta_pos_y = 0
92          self.delta_pos_z = 0.3
93          self.start_pos_x = None
94          self.start_pos_y = None
95          self.start_pos_z = None
96
97          self.desired_speed = 0.001  # Desired absolute speed
98          self.max_deviation_x = 0.1  # Maximum allowed deviation in x-
                direction
99          self.deviation_x = 0.0  # Initialize deviation_x
100
101
102     #get the current position and store it in a vector
103     def state_cb(self, msg):
104         pose = np.array(msg.O_T_EE, order="C").reshape(4,4).T
105         self.current_position = pose[:-1, -1]
106
107         # Update start positions if they are not initialized
108         if self.start_pos_x is None:
109             self.start_pos_x = self.current_position[0]  # x-
                    coordinate
110         if self.start_pos_y is None:
111             self.start_pos_y = self.current_position[1]  # y-
                    coordinate
112         if self.start_pos_z is None:
113             self.start_pos_z = self.current_position[2]  # z-
                    coordinate
114
115         # Calculate end positions relative to start positions
116         if self.start_pos_x is not None and self.start_pos_y is not
                None and self.start_pos_z is not None:
117             self.end_pos_x = self.start_pos_x + self.delta_pos_x
118             self.end_pos_y = self.start_pos_y + self.delta_pos_y
119             self.end_pos_z = self.start_pos_z + self.delta_pos_z
120
121     def process(self) -> None:
122         if self.start_pos_x is None: # No initial reading found
123             return
124         else:
125             if not self.is_initialized:
126                 self.post_init()
127
128         self.elapsed_time += self.timesteps
129         if self.elapsed_time >= self.switch_times[self.
                current_action_idx]:
130             self.current_action_idx += 1
131
132             if self.current_action_idx >= len(self.switch_times):
133                 self.logger_srv(False)
134                 print("Done!")
135                 exit()
136
137             print("Switching to action #", self.current_action_idx)
138         msg = self.actions_msgs[self.current_action_idx]
139
140         default_orientation = [1, 0, 0, 0]
```

```
141         # Check deviation in x-direction from the starting position
142         if self.start_pos_x is None:
143             self.start_pos_x = self.current_position[0]  # Assuming x-
                coordinate is the first element
144         else:
145             self.deviation_x = abs(self.current_position[0] - self.
                start_pos_x)
146             if self.deviation_x > self.max_deviation_x:
147                 rospy.logerr("Exceeded maximum deviation in x-
                    direction. Stopping motion.")
148 
149                 # Set force in x direction to 0
150                 msg.wrench.force.x = 0.0
151 
152         self.pub.publish(msg)
153 
154 
155 if _name_ == "_main_":
156     rospy.init_node("controller_node", anonymous=True)
157     frequency = rospy.get_param("~frequency", 30)
158     node = RosControllerNode(dt=1 / frequency)
159     rospy.loginfo(f"Controller node up and running. Running inference
        at {frequency} Hz.")
160     rospy.Timer(rospy.Duration(1.0 / frequency), lambda _: node.
        process())
161     rospy.spin()
```

Listing A.3: Controller Node Script 2

## A.6 Python Code of Measurement Logger for Franka arm

```
1  #!/usr/bin/env python
2  """ Dummy Node that subscribes and logs messages to .csv """
3  import numpy as np
4  import rospy
5  from franka_msgs.msg import FrankaState
6  from geometry_msgs.msg import WrenchStamped, PoseStamped
7  import message_filters
8  import pandas as pd
9  import time
10 from time import strftime
11 from std_srvs.srv import SetBool
12 from franka_example_controllers.msg import PoseWrenchCommand  # New
       line
13 
14 USE_FT_SENSOR = True
15 
16 class Logger:
17     def __init__(self, output_file, msg_frequency = 50):
18         self.state_topic = "/franka_state_controller/franka_states"
19         self.wrench_topic = "/cartesian_impedance_example_controller/
               estimated_external_wrench"
20         if not USE_FT_SENSOR:
21             self.wrench_topic = "/rokubimini_cosmo/ft_sensor/wrench"
22 
23         self.msg_freq = msg_frequency
24         self.output_file = output_file
25         print("Logger subscribed to", self.state_topic )
26         print("Logging to", self.output_file)
27 
28         self.state_sub = rospy.Subscriber(self.state_topic,
               FrankaState, self.state_cb, queue_size=5)
29         self.wrench_sub = rospy.Subscriber(self.wrench_topic,
               WrenchStamped, self.wrench_cb, queue_size=5)
30 
```

```
31          # New lines start
32          self.ref_cmd_topic = "/cartesian_impedance_example_controller/
                pose_wrench_command"
33          self.ref_cmd_sub = rospy.Subscriber(self.ref_cmd_topic,
                PoseWrenchCommand, self.ref_cmd_cb)
34          self.ref_positions = np.zeros(3)
35          self.ref_forces = np.zeros(3)
36          # New lines end
37
38          self.last_call = time.time()
39          self._data = []
40          with open(output_file, "w") as f:
41              f.write("Fx,Fy,Fz,x,y,z,ref_Fx,ref_Fy,ref_Fz,ref_x,ref_y,
                    ref_z\n") # Updated line
42          self.running = False
43          self._srv = rospy.Service("/measurement_logger/start_logging",
                 SetBool, self.enable_service_call)
44
45          self.wrench_msg = None
46          self.state_msg = None
47
48      def enable_service_call(self, data):
49          self.running = data.data
50          self.last_call = time.time()
51          if self.running:
52              print("Logger is now running")
53          else:
54              print("Logger is now stopped. Logged to ", self.
                    output_file)
55          return True, "Logger is now running" if self.running else "
                Logger is now stopped"
56
57      def state_cb(self, msg):
58          self.state_msg = msg
59          if self.wrench_msg is not None:
60              self.combined_sub(msg, self.wrench_msg)
61              self.state_msg = None
62              self.wrench_msg = None
63
64      def wrench_cb(self, msg):
65          self.wrench_msg = msg
66
67      # New function to handle reference command messages
68      def ref_cmd_cb(self, msg):
69          self.ref_positions[0] = msg.pose.position.x
70          self.ref_positions[1] = msg.pose.position.y
71          self.ref_positions[2] = msg.pose.position.z
72          self.ref_forces[0] = msg.wrench.force.x
73          self.ref_forces[1] = msg.wrench.force.y
74          self.ref_forces[2] = msg.wrench.force.z
75
76      def combined_sub(self, msg: FrankaState, wrench: WrenchStamped):
77          if not self.running:
78              rospy.logwarn_throttle(5, "Logger is not running yet.
                    Waiting for service call.")
79              return
80
81          diff = time.time() - self.last_call
82          if diff < 1 / self.msg_freq:
83              return
84
85          force = np.array([wrench.wrench.force.x, wrench.wrench.force.y
                , wrench.wrench.force.z])
86          pos = np.array(msg.O_T_EE, order="C").reshape(4,4).T
87
88          self._data.append([
89              force[0], # fx
```

```
90              force[1], # fy
91              force[2], # fz
92              pos[0, -1], # x pos
93              pos[1, -1], # y pos
94              pos[2, -1], # z pos
95              # New lines start
96              self.ref_forces[0],  # ref_fx
97              self.ref_forces[1],  # ref_fy
98              self.ref_forces[2],  # ref_fz
99              self.ref_positions[0],  # ref_x
100             self.ref_positions[1],  # ref_y
101             self.ref_positions[2],  # ref_z
102             # New lines end
103         ])
104
105         with open(self.output_file, "a") as f:
106             f.write(",".join([str(s) for s in self._data[-1]]) + "\n")
107
108
109     def run(self):
110         print("Logger is up and running")
111         rospy.spin()
112
113 def main():
114     rospy.init_node("measurement_logger")
115     ts_str = strftime("%Y%m%d%H%M")
116     file_name = f"log_{ts_str}.csv"
117     logger = Logger(output_file = f"/home/zrene/catkin_ws/src/
            franka_ft_reader/output/{file_name}" )
118     logger.run()
119
120
121
122 if __name__ == "__main__":
123     main()
124 strftime("%Y%m%d%H%M")
```

Listing A.4: Logger Node Script