Development of a functional and low-complexity robotic hand

Integrating Adaptive Synergy Actuation and Parallel Individual Finger Control

Master's Thesis

Joost van der Heijden

Delft University of Technology

Development of a functional and low-complexity robotic hand

Integrating Adaptive Synergy Actuation and Parallel Individual Finger Control

by

Joost van der Heijden

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday June 19, 2024 at 9:30 AM.

Student number:	4576675	
Project duration:	July 15, 2023 – June 19, 2024	
Thesis committee:	Prof. Dr. C. Della Santina,	TU Delft, Chair
	Prof. Dr. R. Babuska,	TU Delft
Additional experts:	Prof. Dr. C. Piazza,	TU Munich
	Dr. P. Capsi-Morales,	TU Munich, supervisor

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Abstract

The loss of an upper limb significantly impacts an individual's life, directly affecting their ability to perform activities of daily living. Prosthetic devices play a crucial role in aiding amputees' rehabilitation in society, primarily driven by advancements in externally powered prostheses. Although prosthetic devices on the market offer excellent functionality, they come with a high price tag and use complex control algorithms. Over recent years, a noticeable shift towards simplifying prosthetic devices has emerged, often facilitated by soft robotic principles. For example, the adaptive synergy approach has led to devices that are highly adaptable to their environment with a reduced degree of actuation (DoA), thereby improving functionality at low complexity. This thesis explores a novel research direction by combining the concept of adaptive synergy actuation with additional, parallel actuation of individual fingers. The main goal of this research is to assess the viability of using such a parallel adaptive synergy actuation structure for prosthetic hands. We designed a prototype incorporating this actuation structure, with the main design goals of high functionality, low complexity, robustness, and anthropomorphic sizing. The hand, which features a tendon-driven design, has 15 joints, including 14 dislocatable joints and one revolute hinge joint. The entire hand is powered by a single primary actuator, with two smaller additional motors operating in parallel on the index and thumb. We empirically validated the prototype's performance through qualitative experiments, and its performance was compared to that of other prosthetic devices available on the market through quantitative analysis. Evaluation of the prototype revealed promising results, such as its ability to adaptively grasp various functional objects and execute complex tasks. Force measurements revealed performance comparable to devices on the market. The results indicate that this novel actuation principle, with future refinement, is an interesting new approach to increasing functionality with minimal increase in complexity and can offer an excellent alternative to costly prosthetic devices currently on the market, thereby enhancing the accessibility of functional prosthetic devices for individuals with upper limb loss and improving their quality of life.

Contents

1

Introduction

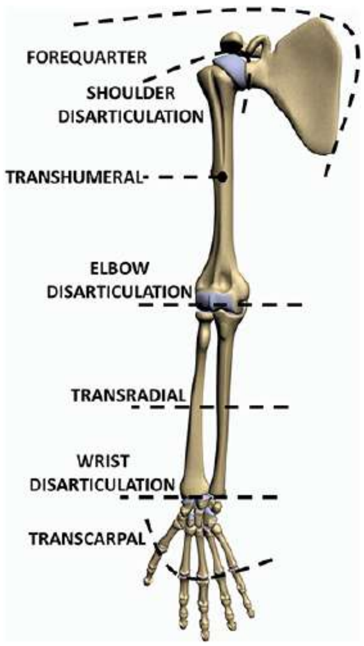


Figure 1.1: Classification of upper limb loss [1]

The loss of an arm or hand can be challenging for an amputee, and the effects directly affect their work life and quality of living. One measure for this is the reduced performance of 'activities of daily living' (ADL). The term ADL refers to routine tasks that comprise everyday living. ADLs are commonly divided into two levels or categories: basic and instrumental tasks. Basic ADLs include mobility and personal self-care tasks. Instrumental ADLs encompass a range of complex tasks that go beyond basic self-care. These activities involve occupational skills, transportation, using technology such as telephones and computers, and household management responsibilities [2]. Trauma is the leading cause of upper limb loss among adults, followed by cancer as the second most prevalent cause [3]. The different levels of upper limb loss can be classified as transcarpal, wrist disarticulation, transradial, elbow disarticulation, transhumeral, shoulder disarticulation and forequarter, see Figure 1.1.

The amputation level is the most critical determinant of function after amputation. The primary surgical principle is to save as much of the limb as possible while ensuring the removal of devitalized tissues and residual limb wound healing [3]. Saving the most distal joint possible dramatically improves the amputee's function. The transradial level is the most prevalent amputation height for upper extremity amputations, accounting for 47% of all upper extremity amputations [4], closely followed by the transhumeral level. Transradial and transhumeral amputees completely lose hand function and have severe limitations in basic and higher-level activities of daily living.

Common solutions

To aid amputees in their rehabilitation into society, intensive care trajectories involving medical, psychological, and social support have been implemented. Often, these trajectories include a prosthetic device primarily aimed at regaining functionality. Based on functionality and general design, prosthetic devices can be classified into several types, namely passive-, body-powered-, externally powered-, and hybrid devices [5].

The names of these types directly reflect their functionality level; as passive devices serve a primarily aesthetic function, body-powered devices are often very durable and can provide basic functionality, such as grasping through mechanical actuation powered by a different body part or limb. Externally powered devices elevate the functionality further by increasing grasping and manipulation capabilities through the use of control algorithms, sensors, and other advanced technologies. Finally, hybrid devices combine body-powered and externally powered principles to create a hybrid device, which aims to provide a balance between the durability of body-powered devices and the functionality of externally powered devices.

Most of the current innovations in prosthetics are pioneered by externally powered devices. Where high discomfort levels often accompany body-powered devices due to the device being attached to various other body parts, externally powered hands only need to be attached to the residual limb, resulting in less discomfort, provided their size and weight are low enough. Externally powered hands come in various kinds, depending on their functionality, complexity, and cost. Nevertheless, a common factor among all types is that they strive to offer high functionality to complete ADLs and achieve high comfort levels. A sub-classification exists between different externally powered prosthetic hands, namely *robotic hands* and *prosthetic hands*. The primary difference lies mainly in their application area and accompanying requirements and limitations. Robotic hands are typically utilized in various industrial processes such as supervised manipulation, autonomous manipulation, and logistics, where higher-order control algorithms can be employed more easily, and size is less of a constraint. In contrast, prosthetic hands are designed for direct human-related purposes, such as rehabilitating amputees. Significantly more straightforward control strategies must be implemented, mainly because of the difficulty in identifying incoming signals and their intended goals. Moreover, prosthetic hands are often strictly limited in size to accommodate high comfort levels.

The key priorities of prosthetic hand design can be divided into two categories: design and functional priorities. The most important design categories are comfort, cost, durability, reliability, and aesthetics. The key functional requirements all stem from how well the hand performs, such as grasping force, available degrees of freedom, ease of control, and object adaptability.

Design priorities

Comfort significantly affects the overall user experience and satisfaction with a prosthetic hand. The weight and size of a prosthetic hand are crucial factors for comfort and ease of use. The comfort of a prosthesis is negatively affected with high weight because a lightweight and appropriately sized hand reduces fatigue and allows for more natural movement. The average mass of a human hand is 426 grams [6]. For that reason, most prosthesis designs set this as the target goal to ensure comfort satisfaction. The price of a prosthetic hand can vary widely, depending on functionality, complexity, materials, manufacturing, research costs, and more. The price ranges from a few hundred euros to tens of thousands. Recent open-source and low-cost projects, such as the Baxter Easyhand, have been launched, claiming prices as low as $150 [7], or Wu's low-cost compact humanoid hand [8]. Contrary, the high-end solutions on the market can cost up to $70,000 (I-Limb)[9], $90,000 (Taska)[10], or $70,000 (Michelangelo) [11]. Additionally, prosthetic hands should be designed to withstand daily wear and tear, providing long-lasting performance. Durability and reliability are crucial to ensure the prosthetic hand functions consistently and effectively over time. In addition, the appearance and aesthetics of a prosthetic hand play a significant role in user acceptance and confidence [1]. Designing prosthetic hands that closely resemble a human hand's natural shape and appearance can positively impact the user's psychological well-being and social integration.

Functional priorities

The ability of a prosthetic hand to generate sufficient grasping force is crucial for securely holding objects of various sizes and weights. Furthermore, adequate grasping force allows individuals to perform daily tasks effectively. Degrees of Freedom (DoF) refers to the number of independent movements or joints a prosthetic hand possesses. Increasing the DoF allows for a broader range of grasp patterns and configurations, enabling users to perform tasks with greater precision and versatility, such as handling objects of different shapes and sizes. The ability of a prosthetic hand to adapt to different tasks, objects, and environments is essential. Designing hands that adjust their grip or shape according to the grasped object enhances versatility and usability. Furthermore, this shape adaptation aids in grasping success by compensating for uncertainties in actuation and world models and attaining many contact points. Thereby shape adaptation significantly increases the chances of achieving force closure with a grasp, and much research on robotic grasping attempts to leverage this effect explicitly [12].

In order to create a prosthetic hand that is valuable to amputees, information is needed on the current state of knowledge, existing technologies, and research relevant to this topic. For that reason, a literature review was conducted on the current state of the art of prosthetic hand technologies. The following section will summarize these findings, highlighting the current trends in the field and identifying challenges, gaps, and opportunities for this thesis.

2

State of the Art

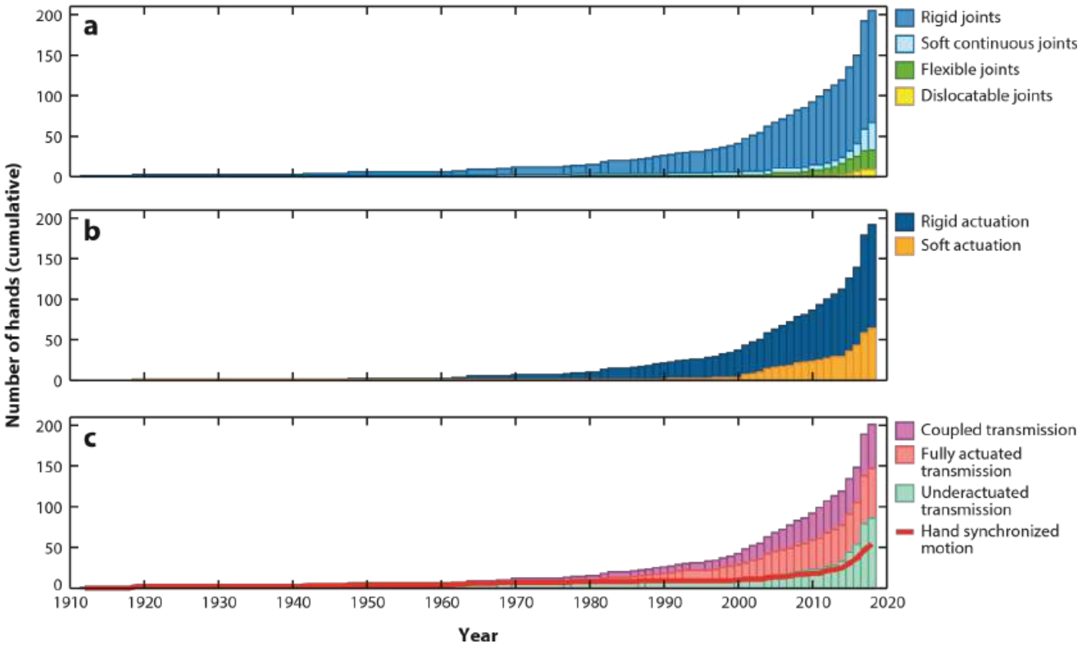


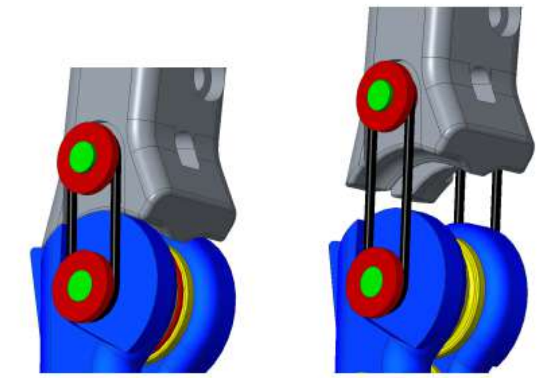
Figure 2.1: Trends in prosthetic hand design [13]. Cumulative distributions of hands from 1912 to 2018 based on the (a) joint type (rigid, soft-continuous, flexible, or dislocatable), (b) the actuation type (rigid or soft), and (c) the transmission architecture type (coupled, fully actuated, or underactuated). In panel c, the red line shows the cumulative number of hands that embed hand synchronized motion

To create a prosthetic hand, it is essential to thoroughly examine all options available for its various parts. The main parts include actuation, transmission, joints, and materials. By exploring different solutions for each component, a knowledgeable approach can be taken to improve the prosthetic hand's functionality and performance. Piazza et al. [13] identified the different trends emerging in three categories for the design of prosthetic hands (see Figure 2.1). Current trends can be easily distinguished from the traditional approach of rigid joints, rigid actuation, and fully actuated transmissions. In all three categories (joints, actuation, and transmission), discoveries have been made that improve overall performance, functionality, usability, and cost-effectiveness. The most notable trends are the 'soft' principles for joints and actuation and reduced degrees of actuation facilitated through novel transmission technologies. The following section will detail these three categories, highlighting the main new discoveries and their benefits and limitations.

2.1. Joint design

Joints in prosthetic hands are crucial because they allow the parts to move relative to each other, enabling more natural interaction with the environment. This mobility is essential for performing various tasks and adapting to different objects, making prosthetic hands functional and effective. Joint design in prosthetics involves various categories broadly classified into four main types: rigid joints, flexible joints, dislocatable joints, and soft continuous joints. Rigid joints are systems where the links are connected using fixed mechanical elements, such as standard hinge joints. Flexible joints incorporate flexible elements (e.g. springs) to connect the links. Using flexible elements creates an inherent compliant effect, which helps with adaptability to various objects. Similarly, dislocatable joints include flexible elements with the addition of withstanding severe disarticulations. One typical example of this is a saddle joint with an added flexible element, which has a concave and convex surface that fits together and is connected with elastic tendons, similar to the joint in the thumb. Unlike the other joint types, which are all made up of multiple parts, soft continuous joints utilize compliant materials and mechanisms to mimic the behaviour of human joints more closely. Soft continuous joints are constructed using materials with inherent flexibility and compliance, such as elastomers, polymers, or composite materials.

Rigid joints are still the most prevalent among all joint types, but a steady increase in the other types is happening. Especially dislocatable and soft continuous joints are becoming more prevalent due to their unique benefits and human-like kinematic behaviour. Two of the most commonly used dislocatable joints in prosthetic hands are the dislocatable rolling contact joints called Hillberry joints [14] and COmpliant Rolling-contact Elements (CORE) [15]. Depending on the design used, such joints can introduce compliance without affecting the precision, making the finger resilient to impacts. Several researchers have made use of this type of joint, such as the Pisa/IIT SoftHand [16], the MeRo Hand [17], and an underactuated hand by Mottard et al. [18], where each finger consists of a group of rolling joints connected by elastic ligaments such as in Figure 2.2. The elastic bands, fixed on either side of the joint, make the system soft and safe and allow the hand to automatically return to its correct configuration, i.e., after severe dislocations. The replacement of rigid mechanical structures with soft materials and actuation has been identified as one of the biggest trends in the design of prosthetic hands [13]. Soft robotic hands exploit the flexibility of joints to adapt the shape of the fingers to the object (or environment) when grasping, substantially simplifying control strategies. The compliant nature of these joints significantly enhances the robustness of the robotic hand, enabling it to withstand significant impacts.



(a) Design of the joints. (b) Joint pulled apart.

Figure 2.2: Dislocatable rotational sliding joint in its two different states used in the design by Mottard et al. [18].

2.2. Actuation

Actuation plays a vital role in the design of prosthetic hands, it being the mechanism that enables the prosthetic hand to produce controlled movements and exert forces as required. Various actuation methods have been explored in prosthetic hand design, each with unique advantages and considerations. Traditional approaches primarily use electric motors such as the ones in Figure 2.3, but pneumatic or hydraulic systems have also been well-researched. Recently, there has been a growing interest in using soft actuators and creating artificial muscles that aim to replicate human muscles. Both aim to incorporate softness into the design and although artificial muscles are still somewhat underperforming, soft actuators have created an interesting way to induce additional compliance and improve performance. Generally, however, soft actuators are significantly more expensive than standard electrical DC motors. A different, often cheaper, approach for implementing softness into a hand's design is by exploiting mechanical advantages in the transmission architecture.

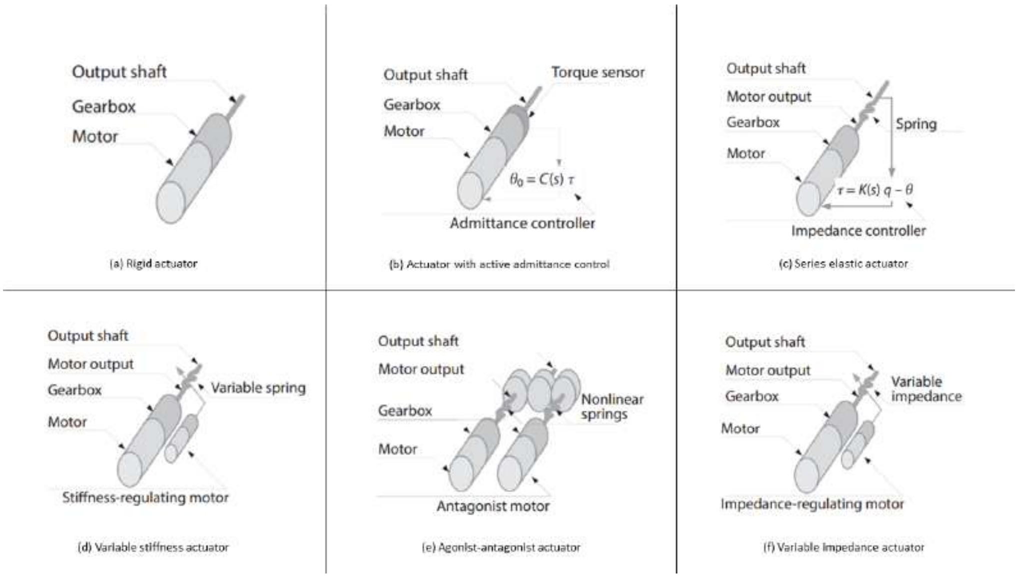


Figure 2.3: Schematic drawings of different actuator solutions: (a) Rigid actuator, (b) Active admittance control, (c) Series elastic actuator, (d) Variable stiffness actuator, (e) Agonist-antagonist actuator, (f) Variable-impedance actuator. Images courtesy of [13]

2.3. Transmission

Transmission systems in prosthetic hands refer to the mechanisms responsible for transmitting motion from actuators to the joints of the hand. These systems dictate how effectively the hand can move and adapt to various tasks. Transmission can be classified into three categories: fully actuated, coupled, and underactuated. Fully actuated transmission systems use a separate actuator for each joint, which means they can perform many different configurations. However, this comes at the cost of a very high degree of actuation (DoA), which requires complex control architectures and comes with high costs. Fully actuated systems are primarily used in high-end solutions for prosthetic hands because they often require advanced sensory input and processing, many actuators, and a sophisticated algorithm to control everything. Coupled and underactuated transmissions have a higher degree of freedom (DoF) than a degree of actuation, meaning a single actuator moves multiple joints. The difference is that the movement of a joint in a coupled transmission system is always directly proportional to a joint linked to it, whereas an underactuated system allows for passive movement between DoFs. This underactuation feature creates a mechanical adaptivity in the hand without the need for complex control algorithms or sensorization. Underactuated hands typically employ specific grasp strategies, such as the synergistic [19] or intrinsic hand model. These models identify key grasping postures or patterns that can be achieved with fewer actuators. The hand's mechanical design is optimized to naturally achieve these grasping postures or patterns. By leveraging passive or self-adaptive mechanical designs and strategic actuation, underactuated transmission systems in prosthetic hands offer a simplified and robust approach to achieving a wide range of grasping and manipulation capabilities. These systems aim to reduce the complexity of control and improve the user experience by leveraging the mechanical properties of the hand itself. Finally, because they use fewer motors, they save weight, space, and cost, which are three important factors in prosthetic hand design.

The transmission of forces from the actuator to joints and moving parts is primarily facilitated by cables and rigid linkages. Vertonghen et al. [20] found that more than half of the devices use a cable transmission. The cable, or tendon, is attached at the fingertip, runs along the finger, and is actuated by a motor-driven pulley. This mechanism is inspired by the tendons of a human hand. The cables or tendons are routed through the prosthetic hand structure in a predetermined manner. They are typically connected to specific points on the fingers or joints to control their movement. Pulleys or guides are often used to change the direction of the cables or tendons, allowing them to navigate through the hand structure smoothly. One of the most significant benefits of using cables is their low weight and small size. Cables can be made from various materials, including steel, Spectra Fiber, Dyneema, and Kevlar. An example of a prosthetic hand using tendons as a transmission form can be seen in Figure 2.4.

(a) Palmar view, fully extended fingers.

(b) Palmar view, partially flexed fingers.

Figure 2.4: Diagram of the internal tendons and pulleys during flexion and extension movements of the middle, annular, and little fingers. (a) Fully extended configuration, (b) partially flexed configuration. 'A-B-C' refers to the pulleys used to connect multiple fingers to a single actuation [21].

2.4. Hand Synchronized Motions

One especially promising trend is the use of a concept called hand-synchronized motions. The study of synchronized hand motions has revolutionized prosthetic hand design, offering insights into how human hands move and interact with objects. Initially proposed by Santello et al. [19], the concept of hand "synergies" revolutionized prosthetic hand design by challenging the traditional view of finger movements. Through extensive analysis of human grasping behaviours, they demonstrated that grasping involves not just the movement of individual fingers but, rather, coordinated movements of the entire hand.

By identifying the patterns of covariation among the 15 joint angles of the hand, they subsequently obtained the coefficients of determination of the relations between the joint angles of the hand. They found a general similarity in the patterns for all subjects, indicating that not all of the joint angles of the hand are controlled independently of each other in shaping the hand to grasp different objects. This implies a reduction in the number of degrees of freedom, and principal component (PC) analysis was used to identify the effective degrees of freedom more precisely. Principal components analysis showed that two principal components could account for >80% of the variance in the data and that the variance contributed by other principal components was small. This result can be interpreted to imply that there are two fundamental synergies governing the manner in which the hand is shaped to grasp objects [19].

Inspired by these findings, researchers have sought to leverage the principles of postural hand synergies in the design of prosthetic hands. One significant advancement in this direction is the concept of adaptive synergies, as proposed by Catalano et al. [22]. Catalano et al. extended the soft synergies framework [23], which was difficult to realize by integrating the viewpoint of soft synergies with that of adaptive underactuated hands, creating the adaptive synergies framework. Adaptive synergies move a step past soft synergies by enabling a method to effectively exploit synergies for the design of underactuated hands, which helps compensate for the reduced number of synergies with the possibility to adapt to the shape of the objects to be grasped.

Another benefit of using adaptive synergies is the possibility of adding more synergies by adding additional tendons in parallel to the first one. A three-fingered gripper prototype noting promising results using multiple synergies has been made [24]. Increasing the number of grasping synergies results in evident benefits, as this enhances the prosthetics' overall dexterity by enabling different grasping patterns. However, multi-synergistic hands require multiple tendons routed independently through all hand joints, which significantly increases the number of pulleys and the overall weight, size, and complexity of the hand. These were the primary limitations of otherwise promising results of [24], making it a promising direction for improvements in future work.

Further advancing the concept of adaptive synergies, Della Santina et al. [25] introduced the concept of augmented adaptive synergies. Building upon the foundation of adaptive synergies, augmented adaptive synergies utilize the inherent friction effects within tendon-driven differential mechanisms to enhance controllability and dexterity by doubling the DoA per tendon by using one motor at each tendon end, with the tendon running through the entire hand. An anthropomorphic robotic hand built using this framework showed excellent grasping dexterity and even manipulation capabilities, proving it to be a novel alternative in the complexity–dexterity tradeoff related to the design of multi-synergistic compliant hands.

The identification of hand-synchronized motion as a highly promising research direction is evident. Most recently, the introduction of augmented adaptive synergies has demonstrated the potential still to be explored in unconventional research directions. Alternatively, the opportunity to incorporate multiple grasping synergies in an adaptive synergies system is promising, too. Adding distinct eigengrasps or fine motor adjustments of individual fingers can significantly increase the overall dexterity and adaptability to various objects. A multi-synergistic approach comes with numerous challenges, such as weight, size, and control complexity, mainly caused by increased actuation and transmission components. As an alternative, adding individual finger control or additional grasping patterns to a synergistic design appears highly feasible, as it takes up much less space than a multi-synergistic approach.

2.5. Design goal

As described in the previous sections, state-of-the-art prosthetic hands have shifted towards enhancing functionality while managing complexity and cost-effectiveness. Dislocatable joints offer improved adaptability and resilience, mimicking human joint mechanics to better interact with varied objects and environments. Meanwhile, traditional electric actuators remain practical for cost efficiency, especially when combined with underactuated transmission architectures. Synergetic design has proven to be cutting-edge technology, although there is still room for improvement. The incorporation of multiple synergies is challenged by factors such as weight and size, resulting in a need for alternative actuation and transmission strategies that could further increase the functionality of these adaptive hands.

Addressing these challenges brings us to the primary objective of this thesis:

Developing a highly functional and low-complexity robotic hand

facilitated through an alternative exploratory actuation principle. This thesis aims to research an approach utilizing a single adaptive synergy to control the entire hand, with two additional individual actuators in parallel to this synergy. The parallel actuators will each move one finger, one being the index and the other the thumb. Other than performing adaptive grasping movements using the main synergy, these two parallel motors can be utilized to move the index and thumb independently from the other digits (such as for pinching) or together with the main synergy, thereby increasing the power output of these primary function fingers.

This novel parallel actuation principle aims to become an alternative approach in the complexity-dexterity tradeoff by increasing the functionality of a low-cost, adaptive synergy actuated hand with minimal increase in complexity. To realize this, a prototype hand was made using the following design goals:

- High functionality
- Low complexity
- Anthropomorphic
- Robust

High functionality is often related to high complexity; however, the aim is to research the dexterous capabilities of a prosthetic hand using the parallel actuation principle. By incorporating the parallel actuators, the hand must have more sensible functionality than using only an adaptive synergy in the design.

Conversely, the hand must be low-complexity, primarily in terms of control and the number and size of the necessary parts. Naturally, there is an increase in complexity compared to a single actuator adaptive synergy design, but the increase in functionality must be more significant than the increase in complexity to prove the viability of the mechanism.

The hand size must aim to be anthropomorphic, as this is the primary limitation of multi-synergistic actuation designs. One of the main complications during the assembly of sophisticated prosthetic hands is tight tolerances because of strict size limitations. Scaling up the dimensions of the hand is generally easier than scaling down, and thus, the aim should be to start at the lower side of average human dimensions to prove the viability of the hand for amputees who lost a small hand.

Finally, robustness is an important design goal. Robust robotic hands must be able to withstand daily use and perturbations from their environment while maintaining their functionality. Therefore, the design will incorporate features that enhance its robustness and not negatively impact its precision.

3

Conceptual Design Exploration

This chapter embarks on a thorough exploration of the design journey undertaken throughout this project. First, the design approach will be explained, highlighting key requirements for realizing the hand's design objectives. Additionally, the prototyping and iterative design processes will be introduced, explaining the refinement process of all components to their eventual final stages.

Subsequent sections delve into specific components, starting with an exploration of the finger design. This section will show the evolution of finger phalanges, highlighting their development from basic working principles to their intricate final designs. Following the design of the digits, the focus will shift to the thumb's design, with particular attention to the joint connecting it to the palm. After discussing the individual fingers, the integration of those fingers into the palm will be shown. This includes the strategic placement of the fingers, the routing of actuation within the palm, and the positioning of all motors utilized for the actuation of all the fingers.

Finally, the chapter will delve into the details of the system control design, providing insights into the electrical design and control architecture required for realizing various grasping motions and the integration of these features into a finalized design.

3.1. Design approach

A first general plan for the design of the hand can be seen in Figure 3.1. The design largely follows the anatomy of the human hand, which has four fingers made up of 3 phalanges (proximal, intermediate, and distal). Meanwhile, the thumb is simplified compared to a human hand. The palm of the prosthetic hand is fabricated as one rigid piece, while a human hand uses metacarpal phalanges for all 5 fingers. Although the 4 metacarpal phalanges of the digits are quite accurately represented as one rigid part, the thumb's metacarpal phalanx ensures its opposition capabilities. By simplifying all metacarpal phalanges into 1 rigid part, the prosthetic hand will lose a DOF. This will, however, significantly reduce both the complexity of the design and its motion control.

To reduce the design and production time of the digits, all digits are made up of identical phalanx parts, where the proximal (PP) and intermediate (IP) phalanges are identical for all fingers, and the distal phalanges (DP) are the same across all five fingers.

All the interphalangeal joints will be rolling contact joints, as these joints accurately reproduce the motion of a human finger joint, with the additional benefit of being a dislocatable joint. As part of the simplification of the thumb metacarpal phalanx, the MCP joint of the thumb will be simplified to a revolute joint.

The actuation layout can also be seen in basic form, where a combined actuation connects all fingers to a single actuator with additional parallel actuation routes for the index and thumb.

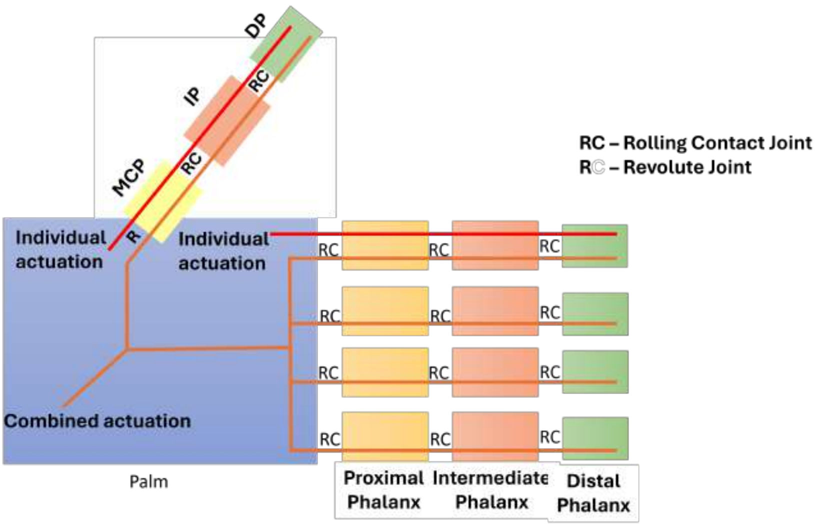


Figure 3.1: Illustration of the hand model featuring the proximal (PP), intermediate (IP), and distal (DP) phalanges. The joints, denoted as RC (rolling-contact) and R (revolute), can be seen in between the phalanges and the palm, and the combined and individual actuation routes are shown in orange and red, respectively.

3.1.1. Prototyping

All plastic parts were produced using FDM 3D printing machines. The palm parts were printed using a Stratasys F170, using ABS material. All other parts were printed using a Prusa MK3 printer, using standard PLA material. In theory, it is not needed to split these parts, the Stratasys machine results in higher quality prints, but at the cost of longer printing times and lower print customizability. In the end-product figure, the ABS and PLA can be easily distinguished due to the use of two different colours, namely white for ABS and black for PLA.

The pulleys used in the transmission of the actuation were primarily printed using PLA material, except the motor shaft pulleys, which were crafted from aluminium using CNC techniques to ensure higher load capabilities. This distinction is important as the areas with aluminium pulleys experience the highest

Figure 3.2: Illustration of the Whoopie Sling [26]

loads. If plastic pulleys were used in those locations, there would be a risk of tendon entrapment or pulley breakage, as the tendon might cut into the plastic during movement, hindering the smooth operation of the system.

The tendon used in the final product is made out of Dyneema. Dyneema is a high-strength synthetic fibre known for its exceptional strength-to-weight ratio. In addition to its excellent strength-weight characteristics, Dyneema rope allows for the use of splicing techniques. Splicing is a useful technique to tie the ends of the tendon into itself, in contrast to a conventional knot. This technique results in not only a stronger connection but also a cleaner and streamlined end result. The specific splicing technique used is known as a 'Whoopie sling', and an example of this can be seen in Figure 3.2.

Like many prototypes, the final product was created through an iterative design process. Multiple iterations of parts were made until all individual parts performed to satisfaction. While not all iterations will be showcased for the sake of brevity, specific refined parts will be presented to highlight the iterative design process.

3.2. Finger design

The finger phalanx design is nearly identical for all the phalanges. The middle, ring, and little finger are all made exactly the same. The index finger works entirely on the same principles, with the added change of allowing for two separate actuation routes.

3.2.1. Finger segments

Both the proximal and intermediate phalanges will be constructed the same way. A single phalanx is made up of two parallel mirrored parts, with space in between them to allow for the tendon routing.

3.2.2. Finger joints

As mentioned in the introduction, the joints used in the finger joints will be rolling contact joints based on the design of the Hillberry joint [14]. The principle of a rolling contact joint relies on two constraints that combine to form a 1-DOF joint. The first constraint is rolling-without-slip, which can be implemented using friction, gears, bands or cables. The second is a normal-contact constraint. This ensures that the segments do not separate at the point of contact. This can be achieved through the use of elastic elements such as springs or cables, such as in the Hillberry joint, or even using links. Using links, however, would impede the desire to create joints that are dislocatable. Because the actuation is transmitted using tendons, the force can only be transmitted in one direction. This means a counterforce is required as a resistance or return system. Using elastics is, therefore, the ideal solution, ensuring all constraints for the rolling contact joint are met and facilitating the fingers' return to their zero state when removing the actuation force.

In Figure 3.3, a visual representation of a single rolling contact joint can be seen. The figure displays the initial joint configuration, considered the zero configuration when local coordinate frames share the same orientation and a different configuration with an angular displacement. A subscript convention is used, where the first subscript indicates the link on which a variable is defined, and the second indicates the adjacent link with which it forms the rolling contact joint, e.g. the radius of the profile on link zero is denoted as r_{01}, and the radius of the profile on link 1 is denoted as r_{10}, highlighting their contact.

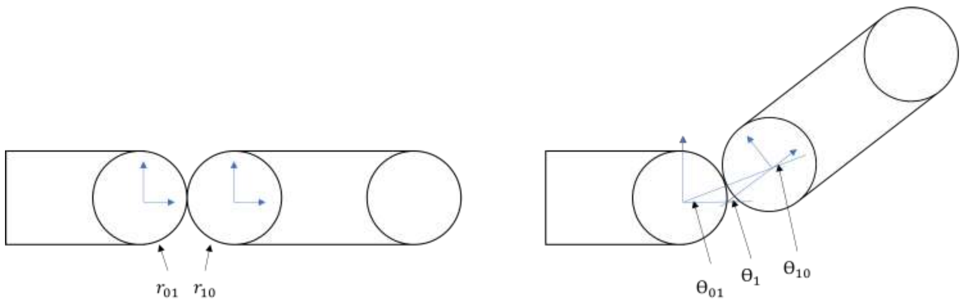


Figure 3.3: Single rolling contact joint before and after an angular displacement

As link 1 rolls without slipping on link 0, the point of contact moves. The relationship between the angle defined by the motion of the point of contact and the relative angle between the links can be determined. The angle of the motion of the point of contact is denoted as angles θ_{01} on link 0 and θ_{10} on link 1, and the orientation of link 1 with respect to link 0 is denoted as θ_1. The arc length defined by the moving point of contact is the same for both links due to the set constraints. This can be expressed as

$$r_{01}\theta_{01} = r_{10}\theta_{10}. \tag{3.1}$$

Analyzing the triangle components, it becomes evident that

$$\theta_1 = \theta_{01} + \theta_{10}. \tag{3.2}$$

Combining Equation 3.1 and Equation 3.2, the relationship between the relative angle between the links and the angles defining the arc length of the rolling motion is found. If one of the angles is known, the others can be found using:

$$\theta_1 = (1 + \frac{r_{01}}{r_{10}})\theta_{01} = (1 + \frac{r_{10}}{r_{01}})\theta_{10} \tag{3.3}$$

The resulting coordinate transformation between the two links is therefore a simple series of transformations which can be written as

$$^{01}\boldsymbol{T}_{10} = \boldsymbol{R}(\theta_{01})\boldsymbol{P}(r_{01} + r_{10})\boldsymbol{R}(\theta_{10}) \tag{3.4}$$

where $\boldsymbol{T}$ denotes the transformation between the frames, $\boldsymbol{R}()$ denotes a pure rotation in the plane, and $\boldsymbol{P}()$ denotes a pure translation in the local frame.

For a single rolling contact joint, the forward kinematics formula is given by Equation 3.4. This accurately describes the position and orientation of the tip of the link. The same logic can be applied to a longer chain of links, such as the one in Figure 3.4. Where the links from left to right represent the palm, PP, IP, and DP, respectively.

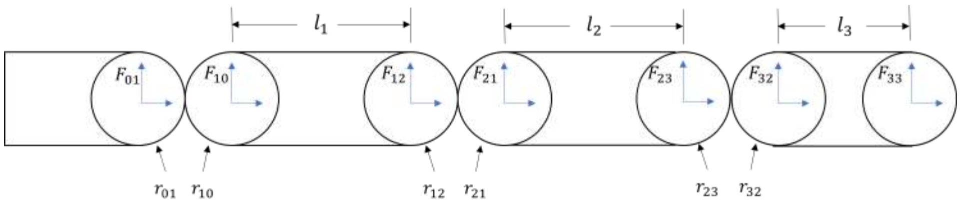


Figure 3.4: An illustration depicting the initial configuration of a four-link, three-joint chain representing the finger segments.

For this kinematic chain, the transformation from the fixed frame F_{01} to the moving frame F_{33} is described by

$$^{0}\boldsymbol{T}_3 = \boldsymbol{R}(\theta_{01})\boldsymbol{P}(r_1)\boldsymbol{R}(\theta_{10})\boldsymbol{P}(l_1)\boldsymbol{R}(\theta_{12})\boldsymbol{P}(r_2)\boldsymbol{R}(\theta_{21})\boldsymbol{P}(l_2)\boldsymbol{R}(\theta_{23})\boldsymbol{P}(r_3)\boldsymbol{R}(\theta_{32})\boldsymbol{P}(l_3) \tag{3.5}$$

where $r_1 = r_{01} + r_{10}, r_2 = r_{12} + r_{21}$ and $r_3 = r_{23} + r_{32}$. As mentioned before, to reduce complexity and increase ease of prototyping, the finger segments will be designed more uniformly than an anatomically correct representation of the human finger. The radii of the different segments are therefore identical, $r_1 = r_2, r_1 = r_3$, and the proximal and intermediate phalanx lengths are also equal: $l_1 = l_2$.

Using Equation 3.5, we can use forward kinematics (FK) to accurately estimate what the finger movement will look like. In Figure 3.5, the end-effector (in this case the fingertip) is mapped for an equal joint angle increment in the range $\{0 < \theta < \frac{\pi}{2}\}$. Incrementing all joint angles by the same amount implies that all phalanges move by an equal amount during the actuation of the finger.

In this scenario, the force applied to the tendon at the fingertip is transmitted through the finger segments with minimal resistance due to the joints' rolling contact nature. Additionally, since the tendon routing is consistent throughout all segments and joint stiffness is considered the same, this contributes to a more uniform distribution of forces and angles across the finger segments.

In practice, the finger is unlikely to behave perfectly, so to adjust its real-life kinematic behaviour, the joint stiffness between each segment can be changed by changing the pretensions of the elastics. This ensures that unwanted behaviour, such as that caused by friction or material imperfections, can be negated.

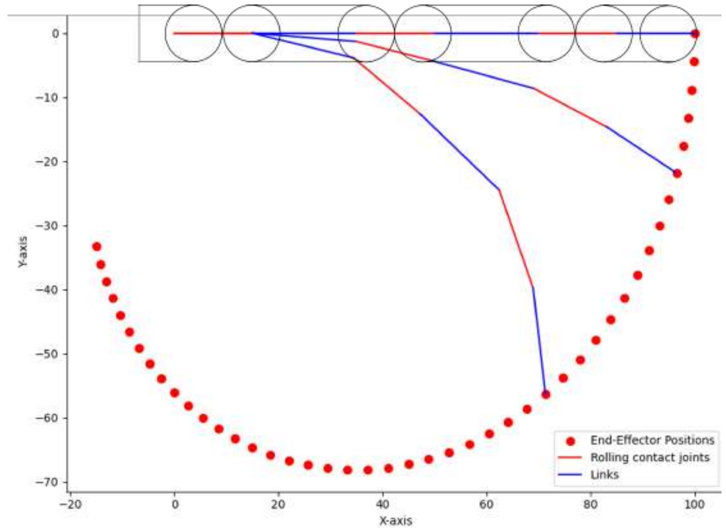


Figure 3.5: End-effector (fingertip) mapping, with intermediary links and rolling contact joints

3.2.3. Finger phalanx design

Now that the basic working principles of the rolling contact finger segments are known, some adaptations can be made to make the finger more robust. As the finger segments only need to move in 1 direction, flexing from the rest position and extending back to the rest position, the movement in other directions can be limited by incorporating physical stops. Using elastics as one of the ways to ensure the contact constraint allows for some play when in its lowest energy state (the rest position). To make the fingers more robust against perturbations in this state and to help guide them back from a flexion state to the rest position, some physical stop will limit the following transformations:

- $+z$ Rotation (Overextension)
- $+y$ Translation
- $\pm z$ Translation

To prevent overextending the joints, the 180° circles can be changed to 90° circles. Changing the top side of the joints to a 'square' is advantageous for several reasons. One benefit of changing to quarter circle joints is that it provides an excellent location to place the elastics, an illustration of which can be seen in Figure 3.6. Additionally, the larger contact surface area allows for implementing another physical stop, limiting the movement in the $+y$ translation direction. Additionally, a possible line of actuation is depicted, which will be further explored in the following section.

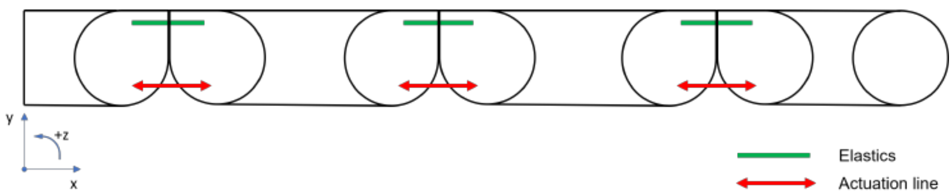
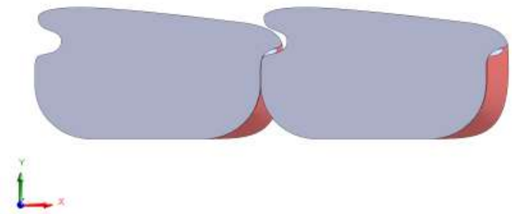
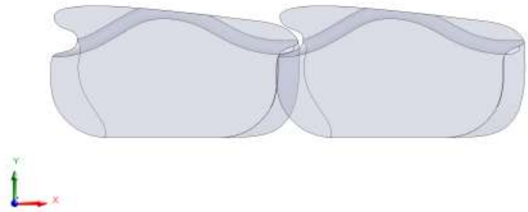


Figure 3.6: Quarter circle joint adaptation with elastic placement and line of actuation

An overview of an implementation containing all additional transformation restrictions can be seen in Figure 3.7. Where extrusions on the front side of the phalanx are made, matching up with cut-outs on the rear side of the phalanx. This creates an interlocking structure that limits the $+y$ and $-z$ translation and the $+z$ rotation. Because the assembled phalanx is made up of mirrored segments, the $+z$ translation is limited by the mirrored parts. The working principle of the rolling contact joint has not changed, as this is still only facilitated by the quarter-circle tangent faces, which are kept intact.

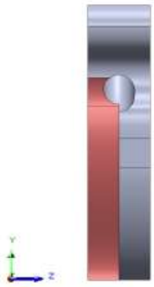


(a) Lateral view of two connected phalanx segments

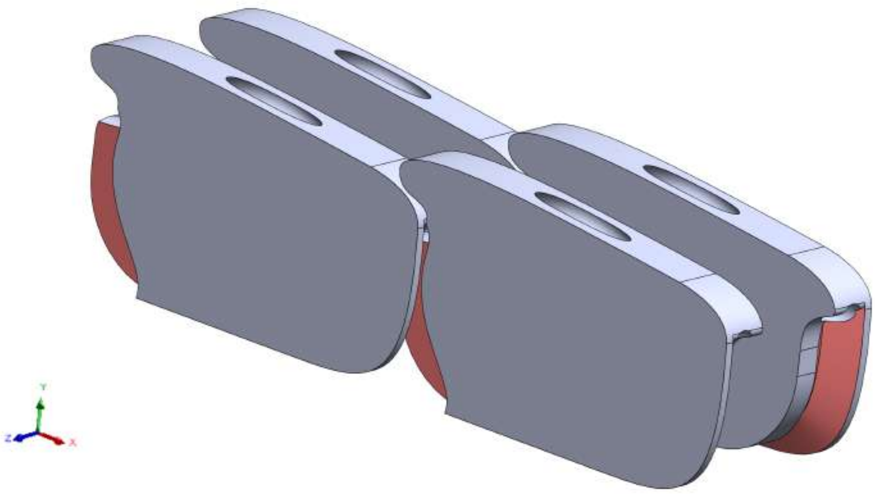


(b) Lateral view of two connected phalanx segments (Transparent)

(c) Front view of single phalanx segment



(d) Rear view of single phalanx segment



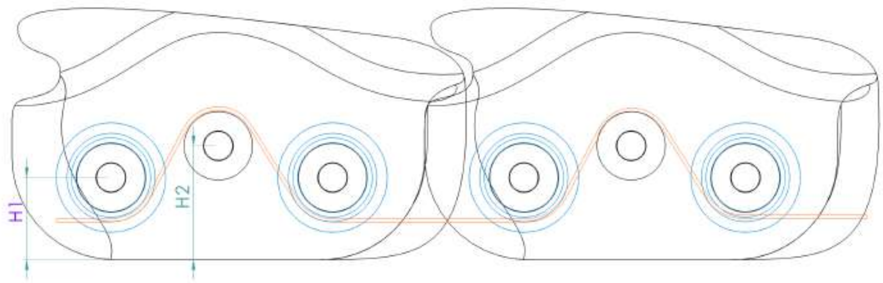
(e) Isometric view of the assembly of two phalanges made up of the mirrored phalanx segments

Figure 3.7: CAD models illustrating finger phalanges from various perspectives: (a) Lateral view of two connected phalanges displaying the interlocking mechanism preventing $+y$ translation and the extrusion and cutout highlighted in red to prevent translation along the negative z-axis. (b) The transparent lateral view of two connected phalanges shows the internal elastic routing at the top and the interlocking mechanism, and the extrusion and cutout (without colour) prevent translation along the negative z-axis. (c) The front view of a single phalanx showcases design details, including the red extrusion as part of the interlocking system. (d) The rear view of a single phalanx highlights the cut-out in red as part of the interlocking system. (e) Isometric view of assembled phalanx segments, forming two finger phalanges. Highlighted in red are the extrusions and cut-outs that, in the assembled configuration, also limit the translation movement along the positive z-axis because of their mirrored configuration

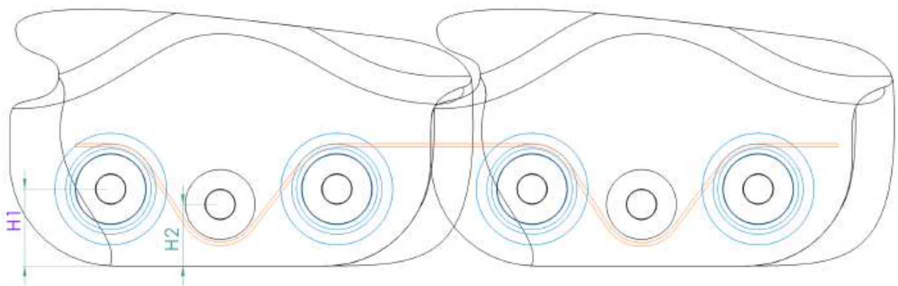
3.2.4. Interphalangeal transmission

The transmission route of the tendon inside the phalanges significantly impacts the outputted force and motion characteristics of the finger. The tendon is transmitted through the finger phalanges using ball bearings and pulleys to minimize friction. The placement of said bearings and pulleys influence the dynamical behaviour of the finger greatly. Placing the actuation line height lower on the segments creates a bigger moment arm around the joint, increasing the force output. However, it is important to ensure the tendon can not protrude outside the phalanges because that might cause it to get stuck or damaged. Different transmission architectures inside the finger segments were empirically studied, as seen in Figure 3.8. The options employ a 'triangle' setup designed to enhance the overall stability of the connection between the two mirrored parts and facilitate the kinematic behaviour of the intermediate phalanx.

The routing configuration used in Figure 3.8a, with variables $H_1 = 6mm$ and $H_2 = 7mm$, demonstrated superior performance and, consequently, is implemented in the final design.



(a)

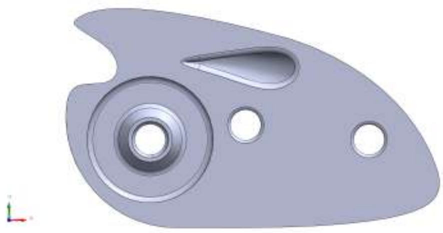


(b)

Figure 3.8: Lateral view of two mirrored tendon routing configurations for interphalangeal actuation transmission: optimization through empirical testing of variable heights (H_1 and H_2) in configurations (a) and (b). The orange line represents the tendon, and the pulleys are highlighted in blue, utilizing ball bearings for reduced friction.

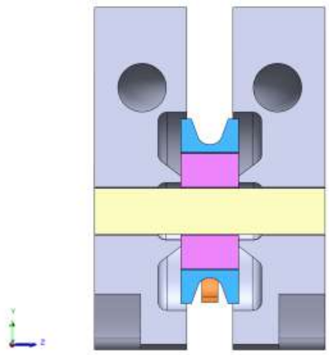
3.2.5. Fingertip design

While the first two phalanges share identical designs, the fingertip has its own set of unique considerations. Unlike the first two phalanges, which have rolling contact joints on both ends, the fingertip, being the end-effector, needs only one. This reduction in joint complexity allows for more flexibility in sizing, enabling adjustments to enhance anthropomorphism. The routing works similarly to the other two phalanges, except that the tendon terminates at the fingertip (see Figure 3.9. Like in the other phalanges, a triangle configuration is employed; however, there is no need for a ball bearing on the second pin, as the tendon remains stationary. Nevertheless, this triangle configuration is necessary to ensure the tendon cannot protrude from the fingertip while moving. A notable difference lies in the routing for the return elastics. As this is the last phalanx, the elastic concludes in a knot. By changing the direction of the elastic routing slightly toward the interior of the phalanx, the knot will be concealed



(a)

(b)



(c)

Figure 3.9: The final version of the fingertip design showcasing (a) a lateral view and (b) an isometric view of one segment, (c) showing a sectioned rear view from the final fingertip assembly. Key features include the droplet-shaped exit hole for the elastic, as well as the cutout and chamfered extrusion around the leftmost pinhole, designed to recess the ball bearing (purple) and pulley (blue) into the segment while retaining low-friction routing of the tendon (orange). The straight pin (yellow) holds the mirrored segments together while also functioning as the axis of rotation for the bearing.

inside the phalanx. The exit hole, resembling the shape of a water droplet, serves a functional purpose in maintaining the secure placement of the elastic. Note also the cutout and chamfered extrusion surrounding the leftmost pinhole. The chamfered extrusion is essential to prevent contact between the ball bearing and pulley with the finger segment. This design feature is crucial because when the edge is chamfered, it only comes into contact with the innermost casing of the ball bearing. This ensures that friction remains minimal, allowing the mechanism to operate smoothly. However, this increases the overall width of the finger phalanx, so to counteract that, a cutout around the pulley is made, recessing the pulley into the segment and streamlining the design. This chamfered extrusion design feature is used throughout the entire hand where ball bearings are used.

3.2.6. Finger assembly

To finalize the fingers, the chamfered extrusion and a surrounding cutout have also been implemented in the two most proximal phalanges. Additionally, the height of the segments has been reduced. Human finger phalanges can be approximated as cylinders, whereas the design of the prosthetic finger phalanges is more accurately represented as unsymmetrical blocks. To increase the likeliness of the designed fingers to human fingers, the height of the top half of the segments has been lowered while leaving the bottom side unchanged to retain its kinematic functionality. By lowering the height and embedding the pulleys into the finger segments, the perimeter of the designed fingers is comparable to the circumference of average human fingers.

As mentioned in section 3.1, the middle, ring, and little finger require a single actuation routing (SAR). The index finger, however, will need 2 actuation routes (DAR). The implementation of this is relatively simple; the two routes use identical configurations and are placed parallel, with a separation part between them. This separation part has a capsule-like shape, which does not protrude outside of the finger segments. Additionally, the separation part uses the same chamfered extrusions as the finger segments on both sides. This ensures that the bearings are aligned and retain their frictionless functionality. The final design of both the SAR and DAR fingers can be seen in Figure 3.10.

Because the additional actuation routing and its required separation part significantly increase the finger's width, the cutouts are deeper than the other digits, embedding the pulleys even more into the finger segments. The total width of the index finger is only 2mm larger compared to the other 3 digits, sitting at 12mm and 10mm, respectively.

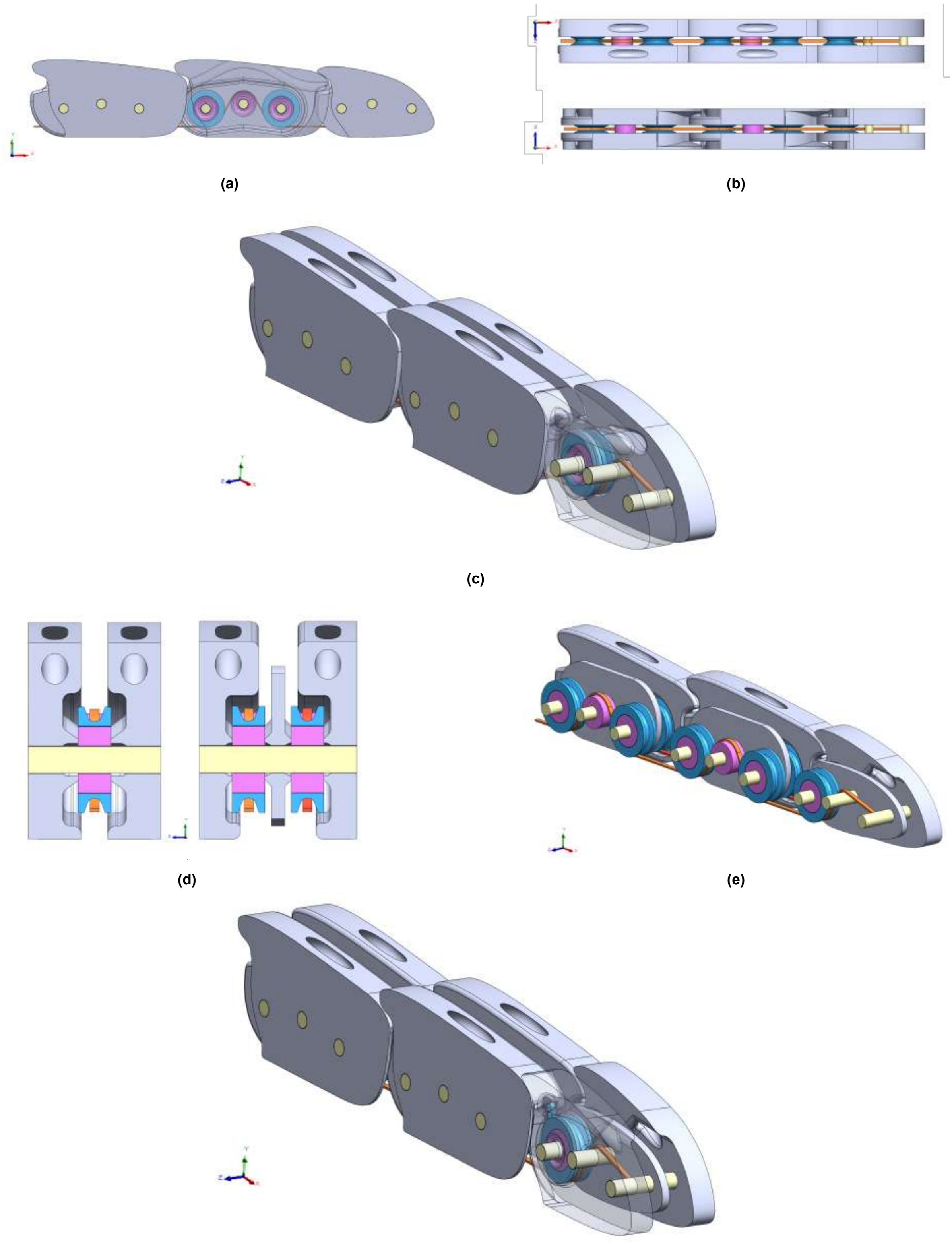
The following table shows average human finger dimensions compared to the dimensions of the designed single and double-routing fingers:

Table 3.1: Comparison of average dimensions: human (adult) finger, single-actuated, and double-actuated fingers (in millimeters)

Finger dimensions	Human (adult) finger [mm]	Single-actuation finger [mm]	Double-actuation finger [mm]
Length	83.0	85.72	85.72
Width	20–25	10	12
Height	x	15.5	15.5
Circumference/Perimeter	50–64	51	55

Table 3.2: Comparison of finger phalanx length of humans and designed single and double actuation finger (in millimeters) [27]

Phalanx	Human [mm]	Designed phalanges [mm]
Proximal	38.0	30.0
Median	23.1	30.0
Distal	21.9	25.7



(a) **(b)** **(c)** **(d)** **(e)** **(f)**

Figure 3.10: The final versions of both the single (SAR) and double actuation routing (DAR) design showcasing: (a) a partially transparent lateral view (SAR), (b) a top and bottom view (SAR), (c) a partially transparent isometric view (SAR). (d) Provides a frontal sectioned view of both fingers, revealing internal structures, the main actuation tendon (orange) and individual actuation tendon (red). (e) Presents a dimetric view (DAR) concealing one side of the finger to reveal the internal structure, showcasing the dual actuation routing design and its separation part. (f) Shows a partially transparent isometric view (DAR) of the index finger. Note the droplet-shaped exit hole in both fingertips for the elastic and the different cutouts surrounding the pulleys and ball bearings. The straight pins (yellow) hold the mirrored segments together to form the finger phalanges while also functioning as the bearings' rotation axis and the tendon's endpoint.

3.3. Thumb design

This section will explore the design intricacies of the prosthetic hand's thumb. Unlike complex counterparts in human anatomy or fully actuated prosthetic options on the market, the emphasis of the designed thumb is on a simplified approach. The designed thumb is made up of three phalanges and three joints. The two distal joints are flexion-extension joints, similar to a human hand, but the MCP joint will be simplified to a 1-DOF joint. Because the prosthetic thumb design prioritizes essential grasping functionalities, the decision to simplify the MCP joint is pragmatic. Time constraints inherent in the prototype development process of this thesis required a compromise between the overall functionality and feasibility of the design.

The design of the thumb tip and proximal phalanx (the thumb does not possess an intermediate phalanx) from the index finger can be reused for the thumb. As the DIP and PIP joints in the thumb are also 1-DOF flexion-extension joints, the same rolling contact joints can be used. Because the thumb is actuated using the synergy and an individual motor, the double actuation routed configuration from the index finger is used. The main difference between the digits and the thumb is the metacarpal phalanx. For the digits, these phalanges are made up of the same part, namely the palm. For the thumb, however, this phalanx will facilitate the opposition-retroposition movement needed for grasping objects. The following section will go in-depth into the design of this phalanx and the resulting final design for the thumb.

3.3.1. Metacarpal phalanx

The design of the metacarpal phalanx for the thumb was a critical part of this project. The prototype's final grasping capabilities heavily depend on this complex part. While using rolling contact joints in the flexion-extension joints used in the fingers is worthwhile, the directional changes needed for the actuation in the metacarpal phalanx would significantly overcomplicate the thumb design if the MCP joint was made similarly. Therefore, the opposition movement will be facilitated by a revolute pin joint.

The metacarpal phalanx's design merges two distinct parts. The distal side of the phalanx facilitates the flexion-extension movement, being one side of a rolling contact joint, whereas the proximal side is used for the revolute joint. Inside the phalanx, the bearings' rotational axes are rotated by 90 degrees to ensure the transmitted forces result in the correct rotational movements of the joints. In Figure 3.11, the thumb design can be seen in detail, showing the assembly together with the phalanges from section 3.2. In Figure 3.11a, the assembled thumb is shown in a slightly flexed state, with the revolute axis highlighted on the green pin (y-axis), which will be secured into the palm in section 3.4. The other subfigures highlight the MCP in more detail from different perspectives. The design idea for the thumb is based on the same idea as the other digits, being constructed out of two mirrored parts with a separation part between the two actuation routes. However, because of the directional changes, one side of the mirror is split into two parts. If it had not been for this splitting, the actual assembly of the MCP would have been impossible. Unfortunately, this setup does impact the structural integrity of the part. After some experiments, additional measures were taken to keep the mirrored parts securely connected, namely adding glue to the straight pins and, more importantly, the addition of a screw for each mirrored segment. These measures ensure that the thumb's MCP can withstand all the forces and perturbations it might encounter.

The separation part used in the revolute part of the MCP has undergone some changes from the other separation parts. The top of the part is widened to allow an elastic to be guided through. This elastic will be attached to the palm and function as the return mechanism for the thumb's abd-add movement. This will become apparent when the thumb is placed in the palm, in section 3.4. It can be seen in Figure 3.11 that two pulleys protrude out of the MCP. Initially, these pulleys were located at the pin closest to the rolling contact joint; however, after some tests, it was found that the thumb performed better in the configuration shown. This was primarily due to irregular misalignment of the tendon in the original configuration, which caused it to get stuck. Another solution to this is adding pulleys to all the bearings. However, this would significantly increase the overall size of the part, and it was therefore deemed undesirable. The primary concern with the protruding pulleys was their possible contact with the grasped object, causing a restriction on the movement. However, this did not significantly impact the final prototype, primarily due to the location of said pulleys and the addition of a glove on the hand.

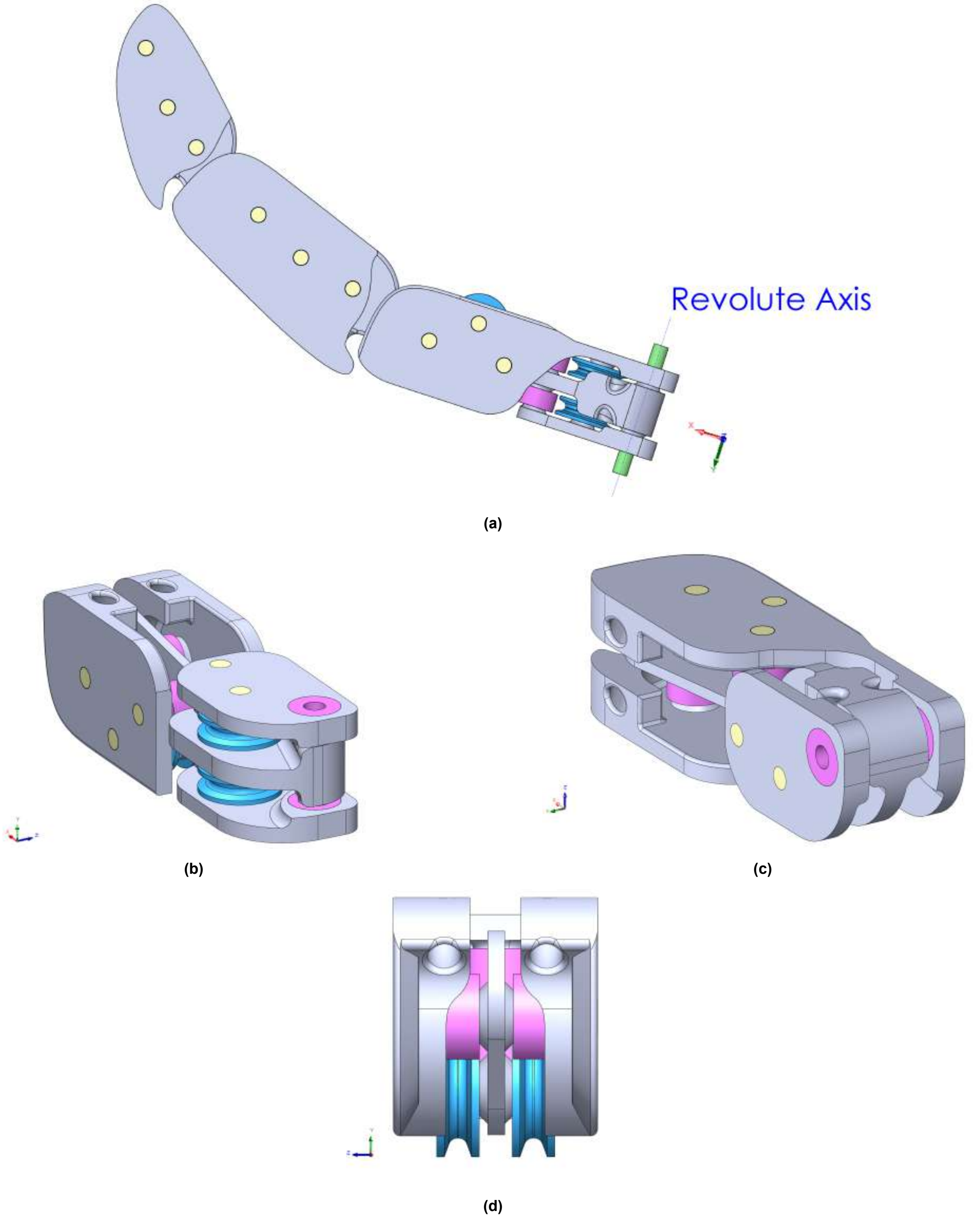
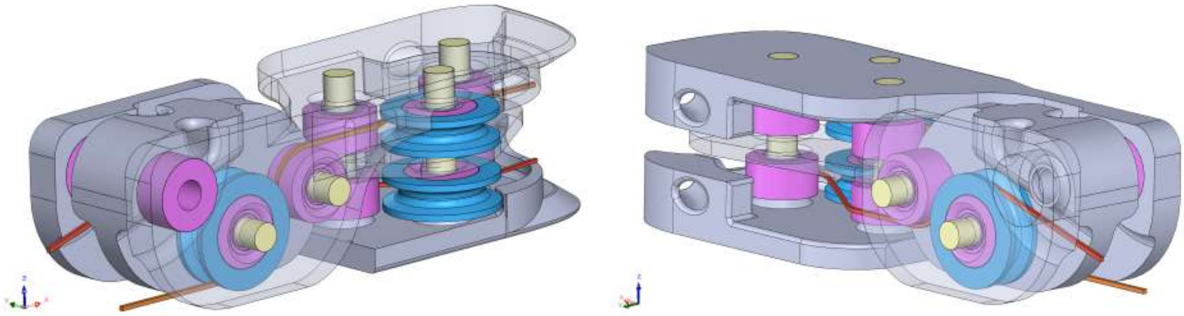


Figure 3.11: Final version of the thumb design showcasing: (a) a dorsal (top) view of the thumb including the revolute axis of rotation as a green pin, (b) an isometric view of the thumb's MCP joint, (c) an isometric view of the MCP joint (rotated 90 degrees from (b)), (d) a frontal view of the thumb's MCP joint showing the flexion-extension rolling contact joint. The straight pins (yellow) hold the mirrored segments together to form the finger phalanges while functioning as the bearings' rotation axis and the tendon's endpoint.

The thumb's actuation transmission adheres to the same setup as the index finger, albeit with slightly more complex routing in the MCP. Two tendons are attached at the fingertip and run through the finger so that all joints follow the desired kinematic behaviour. In Figure 3.12, the main and individual actuation tendons can be seen in orange and red, respectively. The main tendon runs along the dorsal side of the thumb and, through the directional changes in the MCP, ends at the distal side of the thumb (viewed from the wrist). The individual actuation tendon runs along the palmar side and conversely ends at the most proximal side of the thumb (from the wrist). The individual actuation tendon runs over the last pulley, whereas the main tendon runs under the last pulley. This is done to ensure the tendons always remain in tension and contact with the pulleys and bearings, both in the MCP and the rest of the actuation route inside the palm. This will become evident in the following section, where the thumb and the other fingers will be placed within the palm and connected to their required motors.



(a) (b)

Figure 3.12: Tendon routing within the thumb's designed Metacarpal phalanx. The tendon used for the main synergy actuation is depicted in orange (a), whereas the individual actuation tendon is depicted in red (b). The pulleys used to transmit the tendon forces are rotated 90 degrees inside the phalanx to facilitate the correct movement of both the rolling contact joints and the revolute joint.

3.4. Palm design

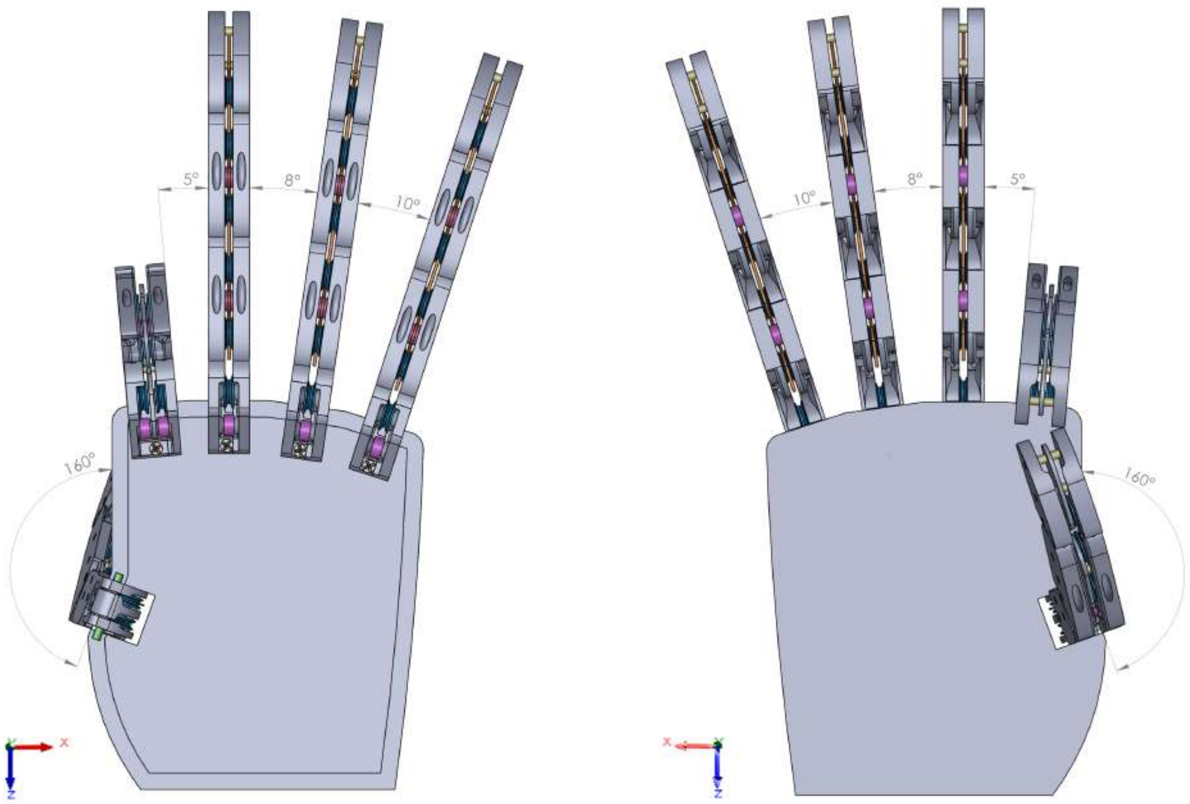
The palm design is an integral part of the entire hand. The placement of the fingers in the palm and the internal routing structure are two key features that make the hand function. Additionally, the placement of the motors that actuate the hand defines its size and the configuration layout for the pulleys that transmit the actuation throughout the hand. The main actuation route used for the synergy grasp connects all the tendons from the fingertips to the main motor, whereas the individual actuation of the thumb and index finger go straight to their respective motors. As these motors are significantly smaller than the motor used for the main synergy, they can be embedded into the palm to reduce the overall thickness of the design. The following sections explain the construction layout of the different layers used to form the palm and the internal actuation structure. Additionally, the placement of the fingers is discussed to ensure the prototype is capable of achieving the desired grasp and pinch movements.

The palm is made up of three separate layers that are connected to form one uniform part. The bottom layer (palmar side) forms the contact surface between the hand and the world it interacts with. Additionally, it houses the transmission connecting the fingers to the main motor and the motors used for the individual actuation of the index and thumb. This layer is connected to a cover part with mirrored versions of all the cutouts needed to clamp the bearings and other parts. This second layer will also provide connection points for the main motor used to move the hand and the required electronics to finally control the hand. Finally, one last part is placed over the main motor and electronic to create a smoother finish.

3.4.1. Finger and thumb placement

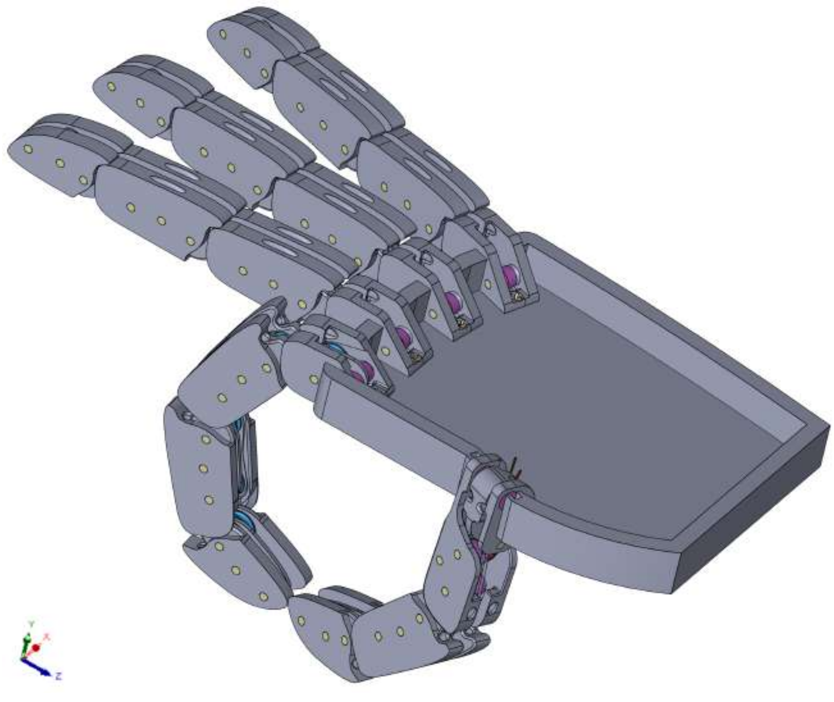
Starting with the placement of the fingers, as for the digits, a connection part between the earlier shown finger segments is screwed into the palm. The orientation of the middle finger is in line with the longitudinal axis of the hand, as viewed from the wrist. The index finger is rotated from the middle finger at a 5° angle to the radial side of the hand, and the ring finger is rotated 8° to the ulnar side of the hand. The little finger, in turn, is placed at an angle of 10° from the ring finger. These orientations are chosen because the designed MCP joint in the digits only allows for 1DOF rotation (flex-ext), disregarding the abd-add capabilities found in human fingers. Although the resting angles (abd-add) of human digits are usually higher (between 10 to 20 degrees), orienting the digits at such a high angle would negatively impact the grasping capabilities of prevalent cylindrical objects. However, placing the fingers at a slight angle increases the grasping capabilities of the hand for objects with non-uniform shapes. Placing the digits at an angle also changes their location in the coronal plane, ensuring the middle finger is the most distal digit, followed by the index and ring finger, with the little finger being the most proximal of the digits. Because of this orientation and location configuration, the hand will have increased grasping capabilities compared to a design with all fingers in a straight orientation at the same distal location in the palm.

The thumb's placement and orientation need consideration of several design requirements and limitations. Because the thumb's MCP joint is simplified compared to a human hand, its work space is also limited. The hand must be able to perform a standard power grasp and pinch. For a pinch, the thumb tip and index tip must line up in a flexed state to create a contact area for objects to be held in to achieve a functional pinch. Because the designs for the index and thumb are already finalised, an assembly can be made of the palm and fingers to determine the thumb's optimal position. In Figure 3.13, an initial design for the palm can be seen that was used for the positioning of the fingers. The thumb orientation is set at 20° from the longitudinal axis of the palm, ensuring that the pinch can be achieved while retaining its more standard power grasping capabilities.



(a) Dorsal view

(b) Palmar view



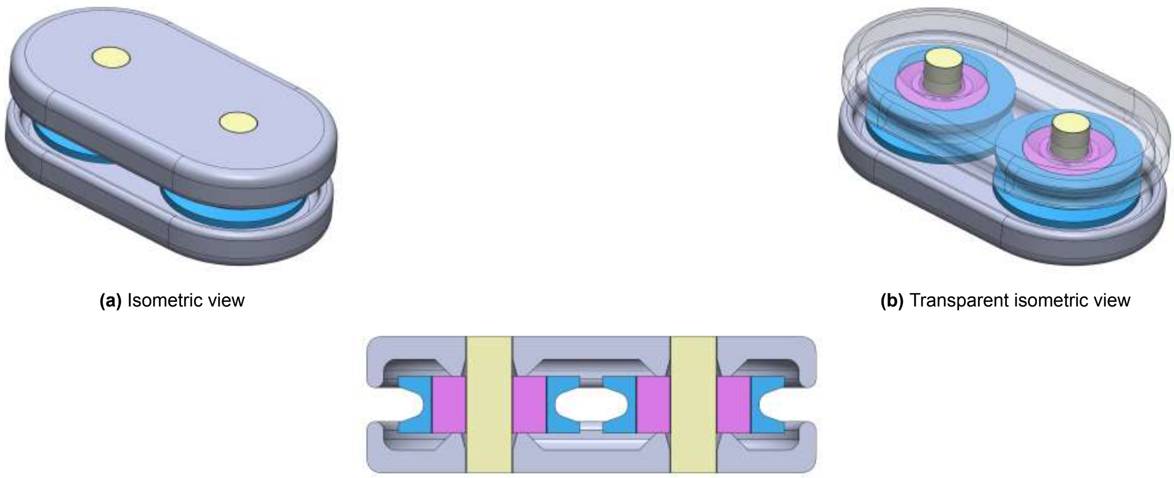
(c) Isometric view

Figure 3.13: Multiple aspects of the finger positioning in the bottom part of the palm. The digits are set at differing angles from the longitudinal axis, which also alters their positions. The thumb is set at a 20° angle to ensure the hand is capable of both power grasps and pinch movements.

Now that all the finger positions are established, the tendon routing and motor placement inside the palm can be finalized. First, the main synergy actuation will be discussed, after which the individual actuation of the thumb and index will be realized.

3.4.2. Tendon routing

As mentioned in section 3.2, the tendons used to actuate the fingers are attached to the fingertips. If every finger uses its own specific tendon, this would leave 5 tendon ends (disregarding the individual actuation tendons) to be connected within the palm. Although this is possible with some clever connection knots and complex internal routing, there is an easier way of achieving synergy actuation. A transmission scheme was created that connects the four digits into two tendon loops. One loop runs from the fingertip of one digit to the one closest to it. These loops are then connected to a different tendon, which runs from the thumb to the main motor, using two 'sliders'. These sliders can be seen in Figure 3.14. The part houses two pulleys that serve as the connection between the digit-loops and the thumb-motor tendon. These sliders will be placed within cutouts in the palm to ensure low palm thickness. The cutouts are dimensioned larger than the sliders, which, combined with the pretension in the tendons, ensures that the sliders are free-floating inside the palm. Ideally, this design choice ensures a minimum increase in friction due to these parts; however, should the sliders come into contact with the palm, these cutouts will ensure they remain level and correctly oriented.



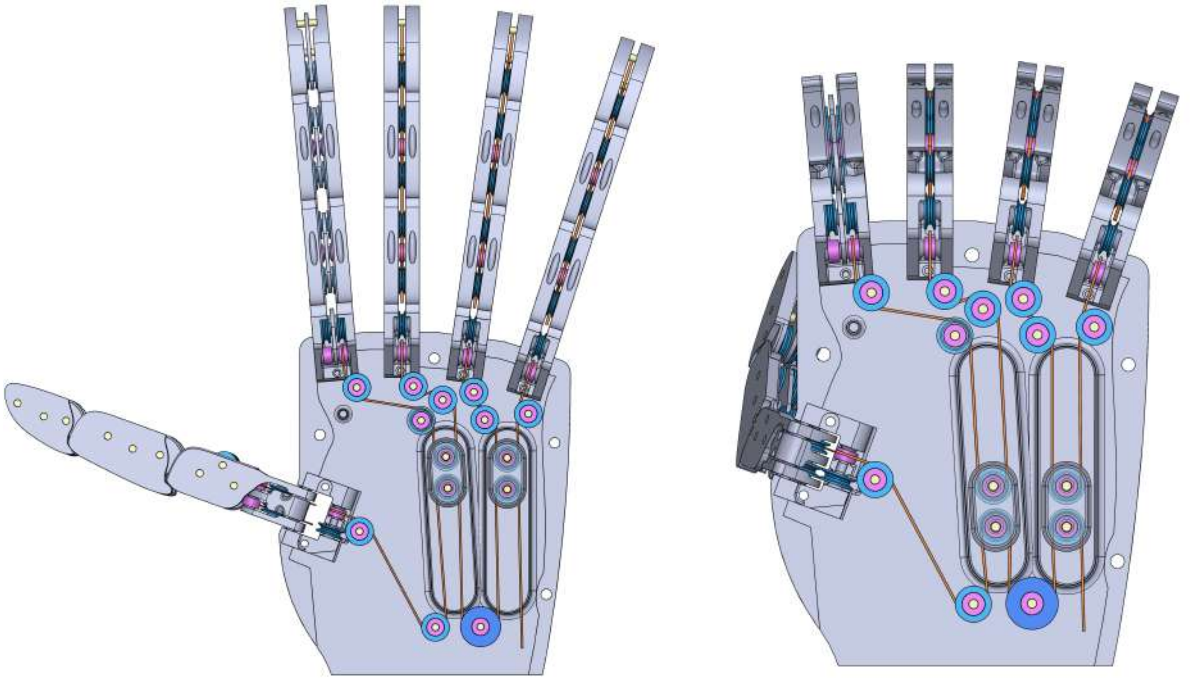
(a) Isometric view

(b) Transparent isometric view

(c) Sectioned side view

Figure 3.14: Isometric (a,b) and sectioned (c) views of the designed slider part. Two mirrored parts (grey) surround two bearings (purple) and pulleys (blue), each connected to one side of the tendon routing.

The placement of these sliders within the palm can be seen in Figure 3.15, where the fingers can be seen in both an extended and flexed state using the actuation routing in the palm. By applying a force at the tendon end that runs from the thumb to the main motor, the thumb flexes and the sliders are moved towards the wrist. Because the two-digit tendon loops are fixed in length, the movement of the sliders flexes the digits simultaneously with the thumb, creating a grasping movement. Although the pulley placement inside the palm might seem strange initially, this is done while keeping the individual motor placements in mind. Because the overall palm thickness can be significantly reduced by placing the motors inside the palm instead of on top, there must be space left to do this. This is the prime reason for making the slider cutouts to the far right, while pulleys are placed specifically to transmit the tendon from the digits to the sliders in a straight actuation line to keep the tendon from dislocating. The pulley used to connect the two sliders (depicted in a darker blue) has a bigger diameter than the other pulleys. The prime reason for that is the increased load on that specific part, as the forces needed for bending every finger are all transmitted over this pulley. Increasing the diameter of the pulley will make it more resistant to wear and friction from the tendon, as it disperses the load over a larger surface area and has more material to begin with.



(a) Fingers extended

(b) Fingers flexed

Figure 3.15: Extended and flexed configuration of the prototype. The thumb will make a flexion movement by applying a force on the tendon running from the thumb. Additionally, because that tendon is guided through the sliders connected to two tendon loops, which in turn are connected to the digits, the remaining fingers will also flex and complete the grasping movement.

3.4.3. Motor placement

The placement of the individual motors can be seen in Figure 3.16. Both motors are placed close to their respective fingers to reduce power dissipation caused by friction. The individual motors benefit more from this than the main motor that actuates the entire hand, as the main motor has a significantly higher power output. The motors used for the individual actuation of the fingers are Pololu High-Power Micro Gearmotors, of which detailed specifications can be found in Table B.1. These motors were chosen because they were readily available, and initial testing on zero-load fingers showed adequate results. Their dimensions allow for placement inside the palm, where small cutouts were made to help secure the motors. The motor used for the index finger had enough space to allow the motor to be wholly integrated into the palm, whereas the thumb motor required some changes to the palm design. The two main reasons are the limited space surrounding the thumb and the necessary alignment of the motor pulley with the pulley guiding the tendon out of the thumb. For these reasons, the motor is elevated slightly so that the motor pulley is placed above the guiding pulley. The transmissions of both the main and individual tendons surrounding the thumb joint can be seen in Figure 3.17. Reflecting on the design choice of running the tendon up and under the most proximal pulley in the thumb in section 3.3, this decision was made in congruence with the palm design to facilitate the individual and synergy actuation routes. By transmitting the tendon in this fashion, the main actuation route is consistently in the same plane, and the individual actuation route remains in tension due to the placement of the motor over the first palm pulley. The same is true for the motor used in the index actuation; because the motor is placed in a cutout in the palm, the actuation line is planar, and as an added benefit, the motor is press-fit in the palm to keep it from moving.

As can be seen in Figure 3.16, the tendon used in the main actuation runs to a pulley in the bottom right of the palm that changes the direction of actuation to the dorsal side of the palm. The tendon will run through the cover layer up to the motor, placed directly above it. This cover layer will be attached to the bottom layer using nuts and bolts, placed along the edges of the palm that have increased surface area to ensure optimal clamping.

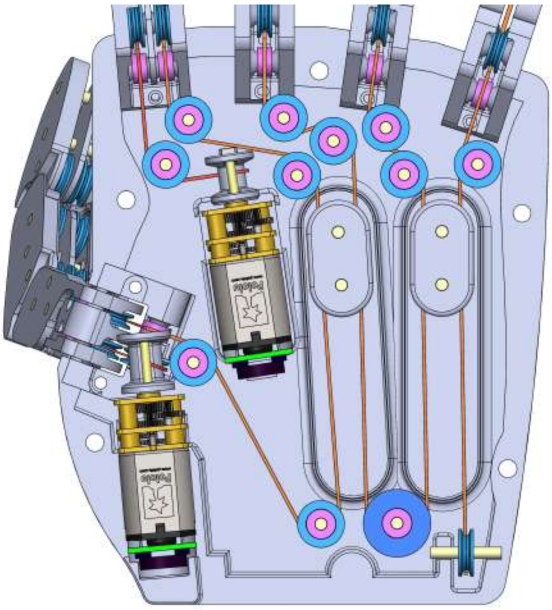
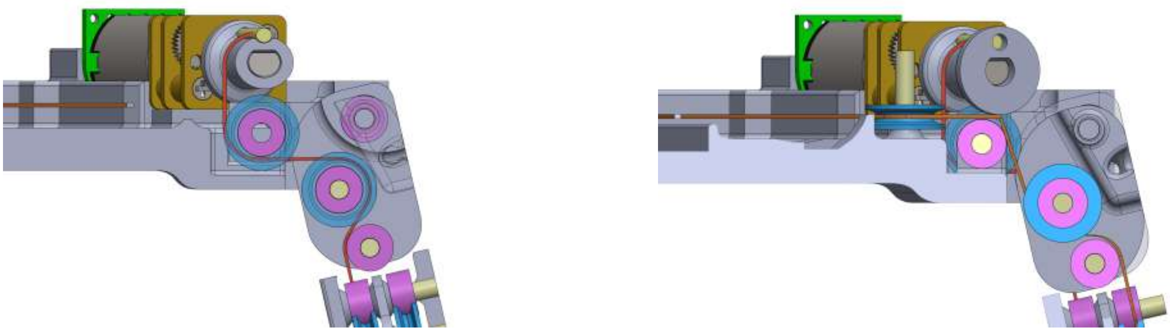


Figure 3.16: Individual motor placement inside the palm ..



(a) Individual tendon routing

(b) Main actuation routing

Figure 3.17: Routing configurations for both the main and individual routing. In (a), the red tendon can be seen routed within the MCP towards the motor pulley, where it is connected to a straight pin. In (b), the main synergy routing can be seen running over a bearing into the palm. Note that the tendon routings use a different configuration to retain tension and help realize the actuation. The motor used for the individual actuation is elevated with respect to the palm so that it can be placed above the pulley that guides the tendon from the thumb into the palm.

The motor assemblies used to actuate the index and thumb individually can be seen in an exploded view in Figure 3.18. The drive pulleys are different for the index and thumb because the thumb has more space than the index. This results in a machined aluminium 4x10mm pulley for the thumb and a 3x8 mm plastic pulley for the index. Both use a pin which runs through the pulley, to which the tendon can attach using a looped knot (whoopie sling). The motors squared off top and bottom allow for custom mounting parts or simply clamping them down, which can be seen when the cover layer is placed in Figure 3.20. The acquired motor assembly can be modified to add a magnetic encoder and magnet to the motor axis. This axis is a direct extension out of the motor and not of the output shaft, which rotates at a different rate due to the 75:1 gearbox. This means that transformations to the encoder data must be performed to calculate the current output position of the motor and accompanying fingers. This will be explained in more detail in section 3.5.

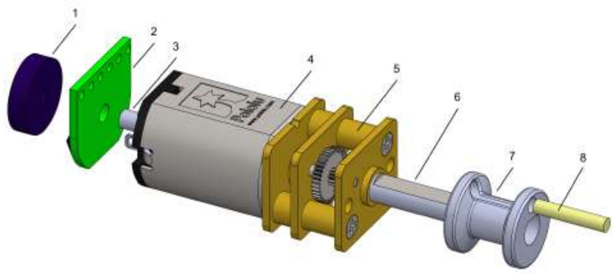


Figure 3.18: Exploded view of the individual motor assemblies. Components include: (1) Magnet, (2) Magnetic encoder, (3) Encoder shaft, (4) Pololu brushless DC motor, (5) 75:1 Pololu gearbox, (6) Motor shaft, (7) Drive pulley, (8) Tendon-pulley pin.

The main motor uses a slightly different approach than the individual ones; see Figure 3.19. Because of the size and shape of the Maxon DCX19s motor used (detailed specifications can be found in Table B.2, a more robust mounting bracket was designed, to which it can rigidly connect with screws. Additionally, the encoder-magnet setup is moved from the back to the front of the motor, connecting directly to the drive shaft. The drive pulley has a centred cutout in which magnets can be placed. The encoder must be placed as centred as possible in front of it and thus has its own mounting bracket, which connects directly to the motor mounting bracket and the palm cover part. This encoder setup ensures that the encoder data can be directly related to the drive pulley's output position, making controlling it more straightforward than the setup used for the individual motors. The final placement of all the motors together with the part that encloses the palm can be seen in Figure 3.20.

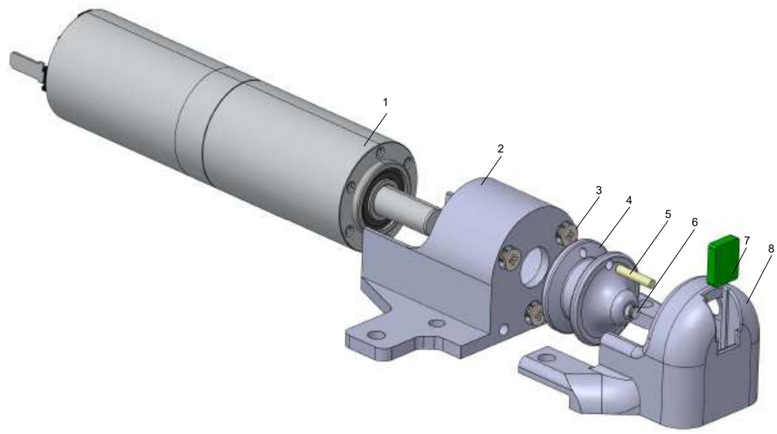


Figure 3.19: Exploded view of the main motor assembly. Components include: (1) Maxon DCX 19s motor, (2) Motor mounting bracket, (3) Motor mounting screws, (4) Drive pulley, (5) Tendon-pulley pin, (6) Magnets, (7) Magnetic encoder, (8) Encoder mounting bracket.

The palm cover encloses the inner transmission with mirrored cutouts for the pins and sliders. A cutout for the index motor is made so the cover part clamps down the motor and keeps it from moving. The cutout for the thumb motor goes all the way through because this motor is placed higher in the plane. Therefore an additional clamping part is made, which slots into the gearbox of the thumb motor, which is connected by screws to the palm to keep it from moving. Note also the two round holes placed directly above the thumb's MCP. These are used to run the elastic from the thumb's MCP to the palm, which functions as its elastic return to the rest position. One of the power pins of the main motor is bent to ensure it remains within the boundaries of the palm, which helps create a uniform shape when the final protection cover is installed. The main motor's drive pulley is placed directly above the pulley that guides the tendon out of the palm. This is necessary to ensure that the tendon does not dislodge and remains within its low-friction pulley setup.

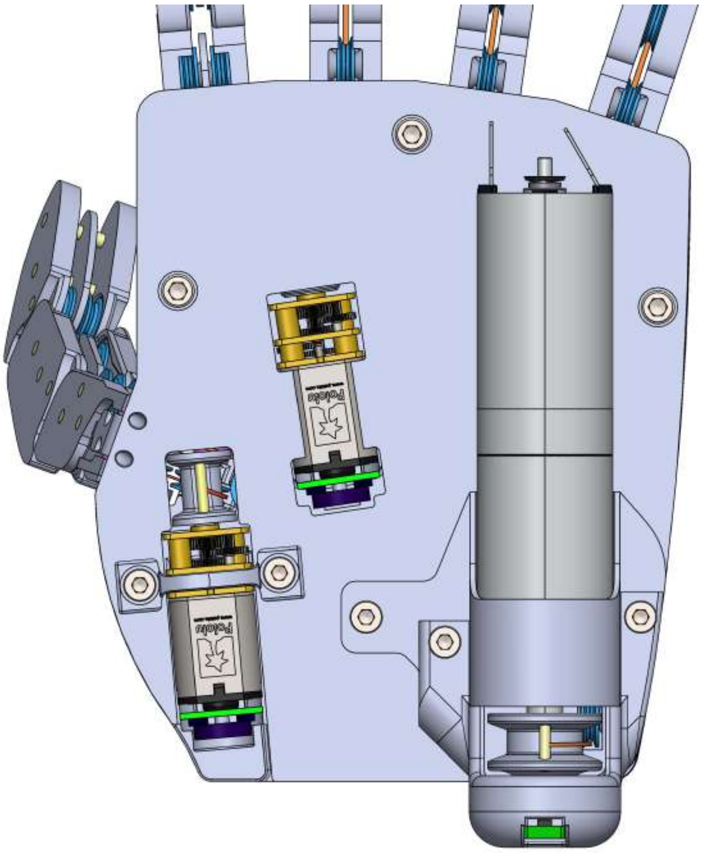


Figure 3.20: Enclosement of the palm and final motor placement within and on top of the cover layer. The thumb motor is placed partly within the palm and secured using a custom 3D-printed bracket and screws. The index motor is fully enclosed by the cover layer acting as the clamping mechanism to secure it completely. The main motor is secured to the palm using its mounting bracket and screws attached both to the cover layer and all the way through the hand to ensure it remains perfectly aligned with the actuation tendon.

3.5. System controller

The system controller enables precise and intuitive control over the prototype's movements. The following section discusses the design and implementation of the system controller for the prototype, which plays a crucial role in its overall functionality. The system controller consists of key components such as a microcontroller, motor drivers, a control station (in this case, a computer), and electronic wiring, all integrated to ensure efficient operation. Having finalized the design of the previous parts of the prosthetic hand, including all the CAD parts, transmission architecture, and motor placement, the focus will next be on the details of the control system architecture. Key elements of the control architecture include implementing a PID (Proportional-Integral-Derivative) control of the motors for precise movement regulation, facilitated by the motor encoders for real-time feedback on motor position and velocity, and establishing the system communication between the controller and the prototype. Subsequent sections will explain in detail the electronic design architecture and the implementation of all required parts into the prototype.

3.5.1. Electronic design architecture

The electronic design architecture is made up of several key components. An overview in the form of a block diagram can be seen in Figure 3.21, where the basic steps taken for creating motion are shown. The command console sends a command in the form of a desired position to the microcontroller. The microcontroller transforms this position command to a pulse-width-modulation (PWM) signal to the motor drivers. The motor drivers use this signal to power the motors, which, when they start moving, send their current measured position back to the microcontroller. The microcontroller adjusts the PWM signal based on the error between the desired and measured positions and keeps doing this until the desired position is achieved. This loop transforms desired motions into the actual movement of the hand, resulting in grasping motions. In the following sections, the microcontroller, motor drivers, command console, encoders, and the integration of these parts will be discussed in detail.

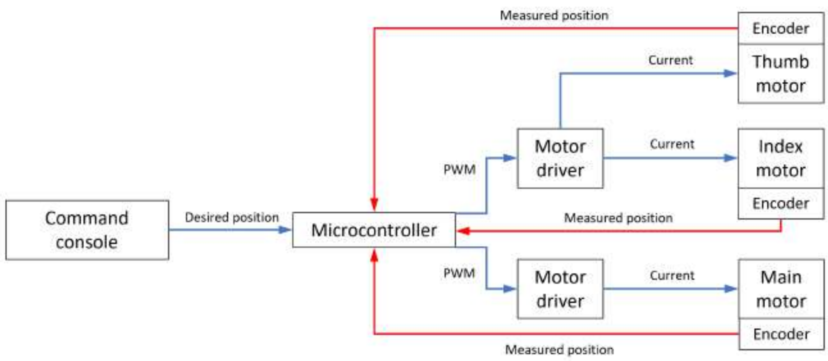


Figure 3.21: Block diagram for motion control. The command console sends a desired position command to the microcontroller, which uses this input and transforms it into motor commands using a PID algorithm. The output command is a PWM signal which is translated by the motor drivers to actually drive the motors, where the motor encoders read their current position which is used as the feedback for the microcontroller to adjust its output signal.

Microcontroller

The microcontroller has the most important function in the electronic design architecture, being responsible for processing the inputs from the command console and sensors (the encoders) and transforming this into actuation signals for the motor drivers. For this project, the ESP-32 microcontroller was selected based on several criteria.

Number of pins

One of the primary considerations in selecting the microcontroller was the number of available pins.

The ESP-32 offers 30 GPIO (General purpose input/output) pins, including analog and digital pins, and an SDA (serial data line) and SCL (serial clock line) pin, necessary to accommodate the different sensors and motor drivers. Not all of these 30 pins are actually useful, as certain pins have specific functions, but the ESP-32 does have all the required pins, so additional processing components are not needed.

Physical size

Another important factor in the selection process was the physical size of the microcontroller. Because one of the design criteria is to keep the size of the hand to a minimum, it would be a great benefit to fit the microcontroller onto the hand compactly. The ESP-32's 58 x 29 mm sizing makes it a very suitable option, given its required number of pins.

Processing speed

Processing speed is crucial for real-time control applications such as for the prosthetic hand prototype. The ESP-32 microcontroller has a dual-core processor with clock speeds up to 240MHz, providing ample computational power to handle the control algorithm, sensor data processing, and communication tasks with the command console. This high processing speed ensures responsive operation of the hand, allowing for precise and natural movement.

As seen in Figure 3.21, the microcontroller is able to transform desired position inputs into a motor movement that matches this input. It does this by using a PID (Proportional-Integral-Derivative) control algorithm. This algorithm is widely used to regulate the output of a system to achieve a desired setpoint (the position). The control algorithm is made up of three terms, each with its own function. The proportional term (P) uses the current error, which is the difference between the desired setpoint and the measured position coming from the sensors. The integral term (I) uses the cumulative sum of past errors over time. The integral term integrates the error over time, which helps correct long-term errors and ensures that the motors reach and maintain their setpoint over time. Finally, the derivative term (D) uses the rate of change of the error. By using the rate of change, the derivative term can dampen the control signal when the error decreases rapidly, which helps prevent overshooting and oscillations.

In a PID controller, the outputted signal is a weighted sum of all three terms, each multiplied with a respective gain coefficient: $K_p, K_i,$ and K_d. These coefficients need to be tuned to optimize the controller and achieve the desired behaviour, which can be done experimentally through trial and error. An example of this in pseudocode is as follows:

```
previous_error = 0
integral = 0
Start:
    error = setpoint - input
    integral = integral + error*dt
    derivative = (error - previous error)/dt
    output = Kp*error + Ki*integral + Kd*derivative
    previous_error = error
    wait (dt)
Goto Start
```

The output value, which is made up of all three terms and their respective gains, can be used to set the motors to a specific power or speed. This is done in the form of a PWM (pulse width modulation) signal. Because the motors used in this prototype are DC motors, they cannot be set at a desired speed directly. For that to work, we need to send a signal to the motors, which translates to our desired behaviour. The PWM signal does just that, as it sends this signal to the motor drivers, which can transform this signal into the desired behaviour. PWM is a method used to control the amount of power delivered to a load (the motors) by rapidly switching a digital signal on and off. The average power delivered to the load is controlled by varying the pulse width while the signal frequency remains constant. In an 8-bit PWM signal, which was used, the duty cycle can vary from 0 to 255, with 255 representing the maximum power output and 0 no output at all. The duty cycle refers to the proportion of time the signal is in the 'on' state compared to the total period of the signal. So, when the desired position is far from the current position, the duty cycle would be set at 255 for 100% ’on’ time, whereas if the current position is close, it would be closer to 64, which equals around 25% ’on’ time.

Motor drivers

Although the microcontroller is great for calculating required outputs, incorporating measured data, and dealing with commands, it cannot supply enough power to the motors to make them work. For that, we need specific motor drivers. The microcontroller outputs the required signal in a low-voltage form (the PWM signal) to the motor drivers, which are connected to a power source, the microcontroller, and the motors themselves. This project uses two different motor drivers: the Pololu DRV8833 dual motor driver for the two Pololu motors used in the actuation for the index and thumb and the Adafruit DRV8871 motor driver, used to drive only the bigger Maxon motor. Both are H-bridge motor drivers, the difference being that the DRV8833 can power two motors at a lower voltage (both 6V), whereas the DRV8871 can only power one motor but at a higher voltage (12V). An H-bridge motor driver consists of four switches in an ”H” configuration. The motor is connected between the two vertical legs of the ”H”, and the switches control the current flow through the motor. By rapidly switching the switches on and off using the PWM signal, the average voltage applied to the motor can be controlled, thereby controlling the power and speed. The wiring diagram for the DRV8833 dual motor driver can be seen in Figure 3.22, connected to the microcontroller, power source and DC motors. The wiring diagram is the same for the DRV8871; however, this one only needs 2 GPIO’s and one output pair to a single DC motor. To further reduce the complexity of the overall system, a single power supply is used for the hand: a Basetech BT-3010, adjustable 0-30V DC power supply. This supply is set at 12V to accommodate the main Maxon motor. A DC-DC converter is then used before the dual motor driver to convert the 12V to 5.3V, which is enough for the smaller Pololu motors.

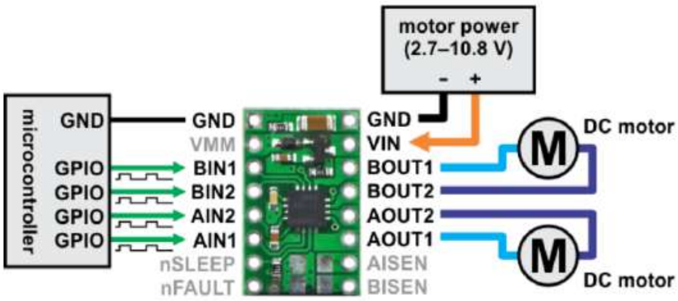


Figure 3.22: Wiring diagram for connecting a DRV8833 dual motor driver to a microcontroller and power source [28].

Command console

The command console acts as the interface between the user and the prosthetic hand. In the case of this prototype, the command console is a very simplistic version of what any future real-life product would use. A computer is connected to the microcontroller using a USB cable, which is used to transmit commands to the hand. As there is no sensory feedback on the position or force of the fingers during a grasp (just the motor position), a GUI (graphical user interface) was made to explore the behaviour of the hand for specific positional inputs. In this manner, empirical evidence could be gathered to complete experiments that validate the hand’s performance, which will be talked about in more detail in chapter 4. The GUI on the command console can be seen in Figure 3.23. The GUI is comprised of four buttons and three text boxes. The buttons are used to start the device, read the current motor positions, stop the device, and finally send the desired positional values to the motors using the three text boxes. The communication runs through the serial socket (USB port), where the microcontroller and GUI can read and send commands to each other as required. As seen in Figure 3.23, the main and individual motor inputs are differentiated as degrees and ticks, respectively. Further explanation of this is provided in the following segment.

Figure 3.23: Graphical User Interface for the control of the prototype. Multiple buttons to start, read current position, stop, and send required positional commands using three text boxes to finely adjust values.

Encoders

As mentioned in the exploded views of both motor assemblies (Figure 3.18 and Figure 3.19), the encoders are placed in different locations in the motor assembly. The Pololu motor-encoder assembly is predesigned in this way, whereas the Maxon encoder assembly is custom-made. The rotary encoders used in the Pololu assembly are very easy to implement in the EDA, as two pins on the encoder can be connected to the microcontroller to count 'ticks'. Each tick corresponds to a small increment of rotation of the motor axis, which can be used to track the motor's position (or speed). The Pololu encoder has a resolution of 6 counts per revolution per revolution of the motor shaft when counting both edges of a single channel (which is the case). To compute the counts per revolution of the gearbox shaft (the drive shaft), we can multiply this number by the gear ratio, in this case: $75 * 6 = 450$. In other words, 450 ticks correspond to a single full rotation of the drive shaft of the individual Pololu motors.

The Maxon motor, however, needs some additional calculation steps. Although the positional data being collected is already directly related to the output axis and pulley, the encoder used is not a rotary encoder as they are not mounted on a shared physical axis. The alignment of the magnets and encoder is, therefore, not as accurate and will require some transformational calculations. The encoder uses a 3D hall sensor, which is able to measure the x, y, and z distance between the sensor and the magnets placed inside the pulley. In this configuration, the z-distance remains constant as this is the rotational axis. The x- and y-distances can then be used to calculate the angle of rotation of the magnets by employing trigonometric functions. Specifically, the arctangent function can be applied to the ratio of the y-distance to the x-distance, yielding the angle of rotation in two dimensions. An example of what this sensory data looks like can be seen in Figure 3.24, where the measured x and y distances can be seen in a 90° phase shift. When using two signals 90° out of phase, it is possible to calculate the angle using the arctangent function $\theta = atan(x/y)$ for the full 360° range. As the alignment of the magnets with the encoder was not perfect, a small offset for both the measured x- and y-distance was required to centre the data around the 0-axis. These offsets were determined by rotating the motor and calculating the respective average data of the x- and y-distances.

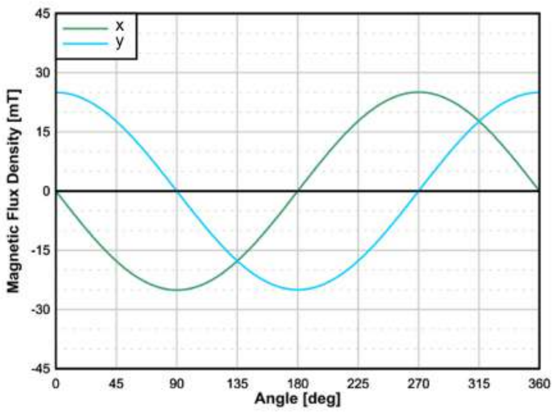


Figure 3.24: Measured x- and y-distance with 90° phase shift. Ideal data with no offset or scaling required for further calculation of angle [29].

By using these trigonometric functions, we can measure the current drive shaft angle in a range from 0-360°. However, this range is too small to perform all required grasping motions. This range needs to be increased to enable full power grasps, where the motor may need to rotate multiple times. This can be achieved programmatically by making use of quadrants. The number of additional turns can be calculated by comparing the current measured angle quadrant to the previous quadrant. The following code snippet was used to do that:

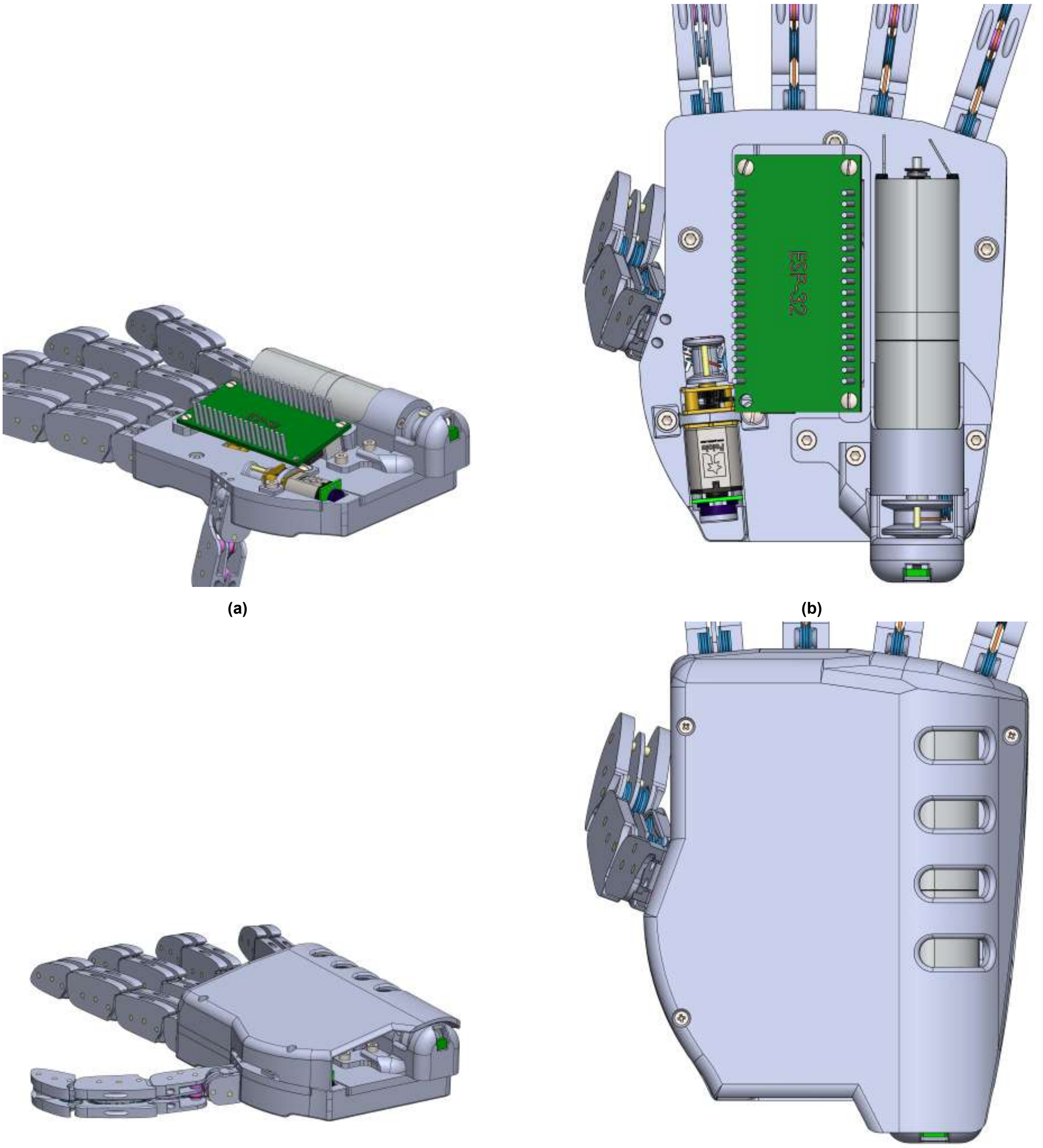
```
void checkQuadrant(float encoderPos)
{
  // Define 4 quadrants (Q) in the 0-360 range.
  if      (encoderPos >= 0 && encoderPos <= 90)   Q = 1;
  else if (encoderPos > 90 && encoderPos <= 180)  Q = 2;
  else if (encoderPos > 180 && encoderPos <= 270) Q = 3;
  else if (encoderPos > 270 && encoderPos < 360)  Q = 4;

  if (Q != oldQ)     // if quadrant has changed
  {
    if (Q == 1 && oldQ == 4)      {numberOfTurns++;}  // 4 --> 1 transition: CW rotation
    else if (Q == 4 && oldQ == 1) {numberOfTurns--;}  // 1 --> 4 transition: CCW rotation
    oldQ = Q;                                         // update to the current quadrant
  }
  // Actual total angle is the number of turns (+/-) converted to degrees, plus the current measured angle within the 0-360 range.
  totalAngle = (numberOfTurns * 360) + encoderPos;
}
```

Using this simple code, the motor's available number of readable rotations is increased, and thus, the motor can finally be used to grasp objects. Before showcasing the hand's interaction with real-life objects, the design will be finalized by integrating the recently discussed required electronic components into the hand.

3.5.2. Integration of electronic components

Now that we better understand the different functionalities of all the electronic components, the parts can be integrated into the design to finalize the prototype. Although attention was given to the size limitations of all parts, repairability and adjustability are even more critical in the prototype phase. Therefore, the motor drivers are kept outside the hand prototype, whereas the microcontroller is fully integrated into the design. The placement of the microcontroller can be seen in Figure 3.25, where the microcontroller is mounted on the palm using a triangular bracket. The microcontroller is placed at an angle, sloping upwards toward the motor. This ensures that the final cover layer can also be sloped, reducing the hand's overall size. The microcontroller is placed above the index motor and is easily accessible by the other motors. The smaller motors both need five electronic wires, three of which run directly to the microcontroller. The remaining two wires are needed for the power and run out of the hand's wrist towards the motor drivers and shared power source. The main motor's power wires also run out of the hand via the wrist, whereas the four wires needed for the encoder run directly to the microcontroller. Finally, to protect all of the electronic wiring and create a smooth, uniform hand, a final cover layer is secured on the hand. The cover layer has four cutouts above the Maxon motor to improve heat dissipation. The cover layer is attached using three screws to make it easily removable for accessing the electronics.



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

Figure 3.25: Finalized assembly of the complete robotic hand. Integration of the microcontroller into the design (a,b) using a custom 3D-printed mounting bracket to reduce the final size of the palm. Final cover layer design in (c,d) to protect the electronics and create a homogenous design that fully encloses the palm.

4

Experiments and Results

In this chapter, the experiments conducted to assess the quality and performance of the prototype will be elaborated on. There were two main goals in these experiments: to validate the functionality and effectiveness of the device and to gather insights for further refinement or improvement. Different experiments were conducted using both qualitative and quantitative approaches. First, the qualitative experiments will be detailed, focusing on understanding the prosthetic hand's behaviour and its interaction with various objects. Next are the quantitative experiments aimed at objectively measuring the specific performance of the hand to directly compare it to other on-the-market prosthetic hands. These quantitative measurements offer empirical evidence to identify the prototype's strengths and possible improvement points. For all experiments, a glove was fitted over the designed hand. This helps increase the friction between the hand and its environment, increasing its grasping capabilities, especially for slippery and thin or small objects. Additionally, it increases the anthropomorphic look of the hand as the plastic components are hidden.

4.1. Finger experiments

This section will focus on the functionality of the individual fingers, showcasing features and capabilities, including individual finger movement of the index and thumb, softness, and quantification of the force exerted by each finger.

4.1.1. Index and Thumb movement

The two motors used for moving the index and thumb can be individually and congruently controlled. This results in three distinct movement scenarios: moving only the index, moving only the thumb, and performing an open pinch (pinching with the three remaining digits in an extended state). The first scenario can be seen in Figure 4.1, where the index finger goes from an extended to flexed state from left to right.

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

Figure 4.1: Photosequence of index-only movement.

The second scenario, moving only the thumb, results in two movements, first the thumb's abduction and second the thumb's flexion. This can be seen in Figure 4.2, where the abduction is visible in the three left frames and flexion in the right two.

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

Figure 4.2: Photosequence of thumb-only movement, including the abduction movement in the first three frames, after which the flexion movement completes the motion.

Finally, when the index and thumb are actuated congruently, the hand performs an open pinch seen in Figure 4.3. The thumb and index fingertips come into contact at a slight angle.

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

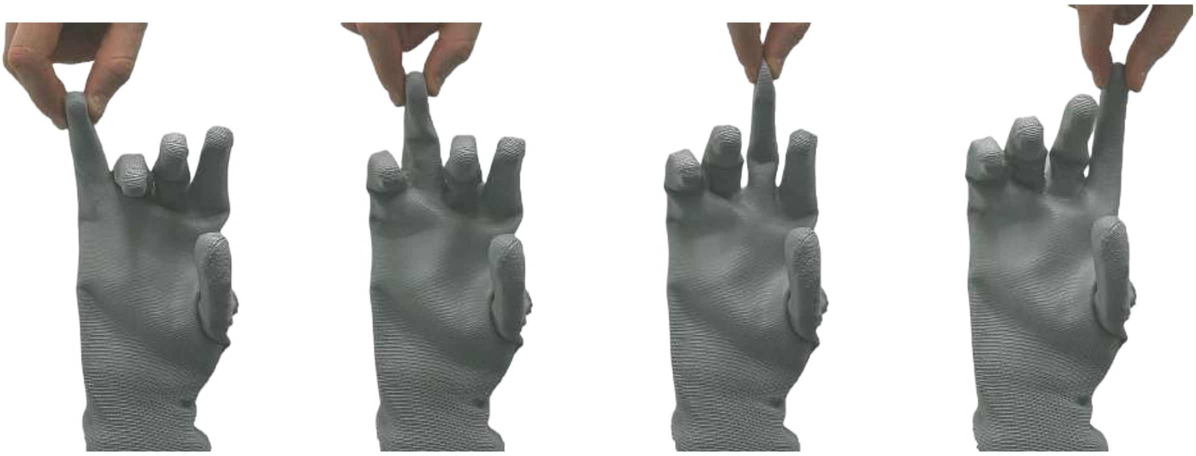
Figure 4.3: Photosequence of open-pinch movement, individual actuation of the index and thumb to achieve pinch movement while other digits remain in resting state.

4.1.2. Soft features and finger interconnectedness

Because of the finger joint design, the fingers exhibit interesting soft behaviour. The joints are dislocatable, which means they are able to withstand perturbations and return to their resting state.

Dislocation figures

Another feature resulting from the underactuated design is the direct finger interconnectedness. This results in adaptable behaviour relative to the object's shape and size, which can also be shown by perturbing a single finger and observing the behaviour of the other digits. As seen in Figure 4.4, when the fingers are moved to a slightly flexed state and perturbations are applied to a single finger, the hand behaves as expected. When one finger is in an extended state, and another finger is also moved to an extended state, the initially extended finger flexes back to the original slightly flexed state. The finger pairs (little-ring and middle-index) display this behaviour most noticeably because the friction losses for a single pair are lower than those of one pair to another.



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

Figure 4.4: Showcase of direct internal connection between the fingers. Each figure displays the next digit being moved, so (b) displays the movement of the ring finger, which results in the return of the little finger; (c) displays the movement of the middle finger, which results in the return of the ring finger and so forth.

4.1.3. Force measurements

The fingertip force was measured for all individual fingers. The forces are measured using a 1-dimensional load sensor. The measurement setup can be seen in Figure 4.5. The load cell measures the force exerted by the fingertips in a flexed state and is aligned with each finger for their respective measurement. A 5 kg load is clamped down on one side, and a force is exerted on the other end. The load cell contains a strain gauge that changes resistance when subjected to mechanical strain. The change in electrical resistance is very small and thus is amplified using a HX711 load cell amplifier. The amplifier is connected to a PC using a USB cable, where an Arduino IDE script reads the measured data and converts the electrical data to force outputs.

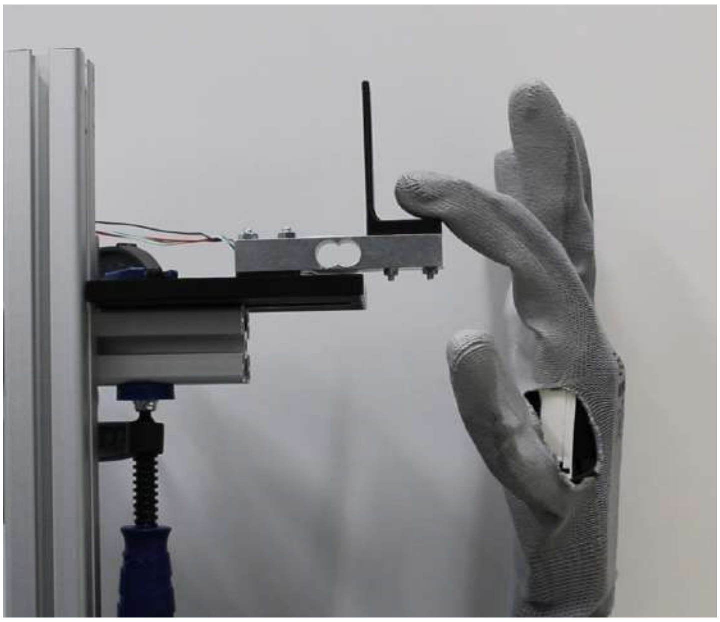


Figure 4.5: Force measurement setup for individual fingers using a 1-dimensional load cell

The fingertip forces were measured using only the Maxon motor, only the individual motor (index and thumb), and both the Maxon and individual motor at the same time (index and thumb). The results can be seen in Table 4.1.

Table 4.1: Fingertip force measurements using three motor setups

	Little finger	Ring finger	Middle finger	Index finger	Thumb
Maxon motor	4.58 N	3.05 N	3.12 N	2.69 N	0.88 N
Pololu motor	-	-	-	1.06 N	2.12 N
Maxon and Pololu	-	-	-	3.31 N	2.53 N

The measured forces vary significantly for each finger, which can be explained by multiple things. The first reason is the measurement setup used for these experiments, which influences the thumb measurement when using just the Maxon motor. Because the hand is underactuated, all of the fingers move when using the Maxon motor. This creates the grasping motion of the hand, which hinders the measurement of the thumb tip in this setup. Because the thumb first abducts and only then starts flexing, the other fingers obstruct a direct connection between the thumb tip and the load sensor. Two other things to note are the individual force measurements when only using the Maxon motor (note the force decreases from little finger to index) and the difference between the Pololu motor measurements. These will be explained in more detail in chapter 5.

4.2. Hand experiments

4.2.1. Functional objects

The prototype can perform different power grasps using only the main actuation motor. The hand's adaptability ensures it can grasp variously sized objects, as seen in Figure 4.6. The positional control command sent to the motor was set at 900° for the three rigid objects, whereas the soft ball used a 1050° command because the object size reduces when applying pressure. The fingers can be seen to interact with the object in a manner specific to the surface it encounters. For example, for the rigid blue ball, the object is relatively small, and as such, the little and ring fingers flex completely, which provides a counteracting force for the thumb to ensure the object remains in the hand. On the other hand, the pear's middle finger is angled because the surface it encountered was initially too steep; however, when the surface flattened, the contact surface and friction increased, and the object was grasped more securely. Lastly, the plastic bottle is grasped using a standard cylindrical wrap, where the fingers grip around the circumference of the bottle with moderate pressure.

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

Figure 4.6: Power grasps of various objects showcasing adaptability to object shapes. Objects from left to right: Rigid ball, Soft ball, Plastic pear, Plastic bottle.

In Figure 4.7, a photo sequence of the steps taken to successfully pinch a cherry can be seen. In figures a-e the first movement of the hand can be seen, where an actuation command was sent to all three motors. The first command ensures the cherry can be held independently by the hand (e), after which the fingers are actuated once more to create a closed pinch movement (pinch with thumb-index with all other digits flexed to the palm). This sequence creates a realistic object approach to a fully completed pinch, which can securely hold the cherry even when perturbed.

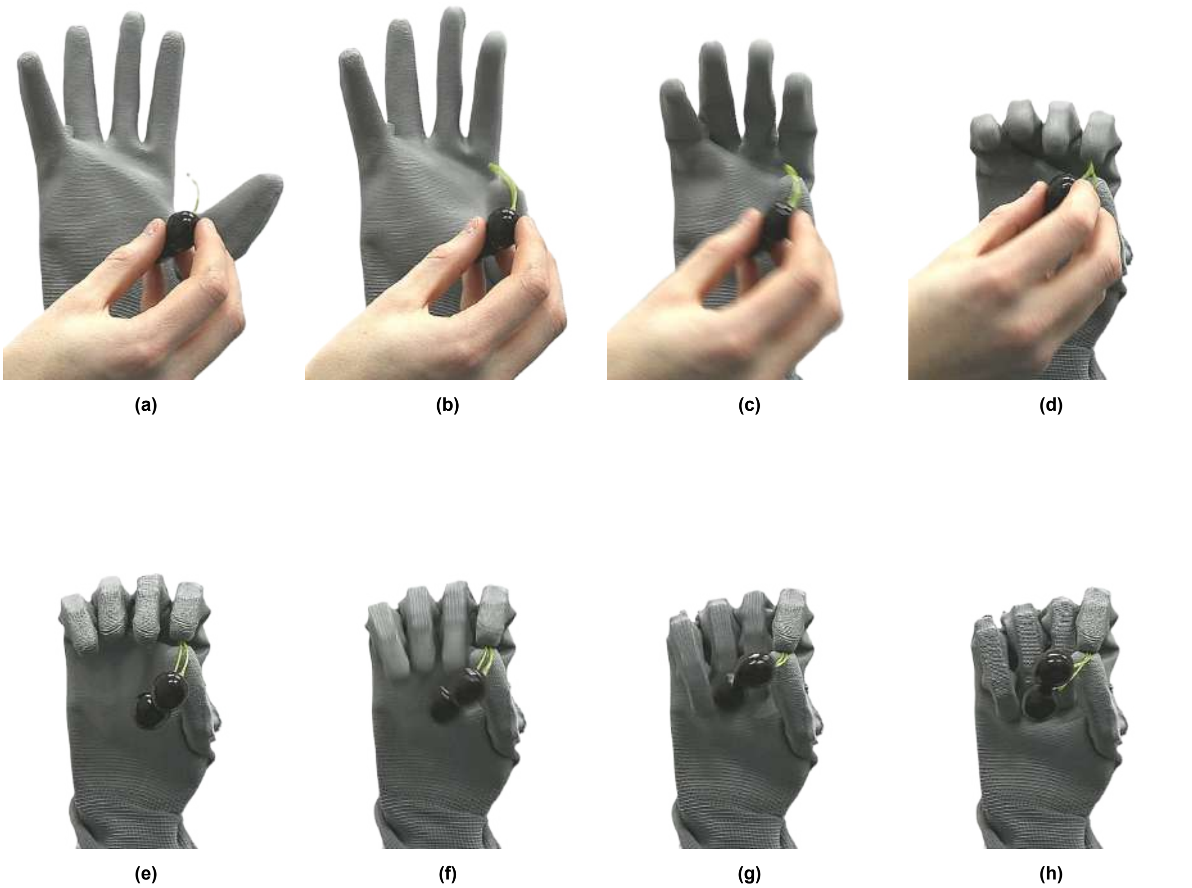
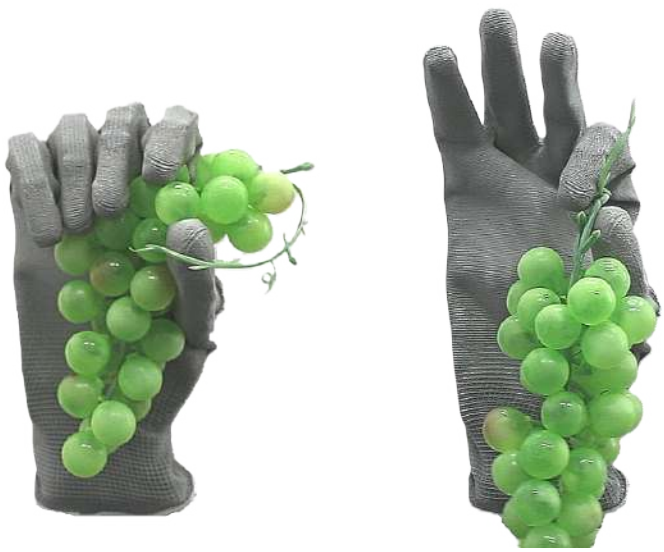


Figure 4.7: Photosequence of pinch grasp on a cherry. An initial approach is made to hold the cherry independently, after which the hand is closed further to complete a fully closed pinch, which holds the cherry securely.

In Figure 4.8, two different approaches to the same object can be seen. In (a), a power grasp using only the main actuation degree is used to hold the grapes. In (b), an open pinch is performed using primarily the individual actuation degrees to create an open pinch that holds the grapes more naturally. These distinct grasping strategies demonstrate the versatility of the prototype in handling delicate objects like grapes (in this case, plastic grapes).



(a) (b)

Figure 4.8: Comparison of two grasping options for the same object (grapes), power grasp (a) is obtained using the main actuation. The open pinch is obtained using a combination of all three actuation degrees.

In Figure 4.9, different pinch grasps can be seen to grasp different objects. In (a), a thin circular disk is held primarily using the individual actuation degrees, with additional support provided by the main actuation degree. In (b), a marker is held using a fully closed pinch; in (c), the remaining digits are not flexed completely while holding a normal pen. Both create a natural grasping option that can be used to utilize the objects.

(a) (b) (c)

Figure 4.9: Grasping objects using open pinch: (a) disk; and closed pinch: (b) Marker, (c) Pen.

Finally, another great example of the hand's versatile object-handling capabilities can be seen in Figure 4.10. An initial approach is made to pinch the rubber duck between the index and thumb, after which a power grasp is done to hold both the duck and a soft ball. This shows both the hand's adaptability to object shapes and the possibility of sequentially grasping two different objects distant from

(a) (b)

Figure 4.10: Sequential grasping of two objects: (a) Pinching the rubber duck to hold it independently by primarily using the two individual actuation degrees, (b) Power grasping the soft ball using the main actuation degree after the original pinch.

each other.

4.2.2. Tasks

In addition to grasping and holding various objects securely, the prototype can also perform tasks with various functional objects. The first task can be seen in Figure 4.11, where a photo sequence shows the use of tweezers to grasp a thin piece of sandpaper. Figures a-e show a closed pinch sequence, where the tweezers are closed step-by-step. In (f), the pinch is opened slightly to open the tweezers and allow for the placement of the sandpaper. In (g), the pinch is closed completely again to fully pinch the tweezers, thereby holding the sandpaper securely and completing the task.

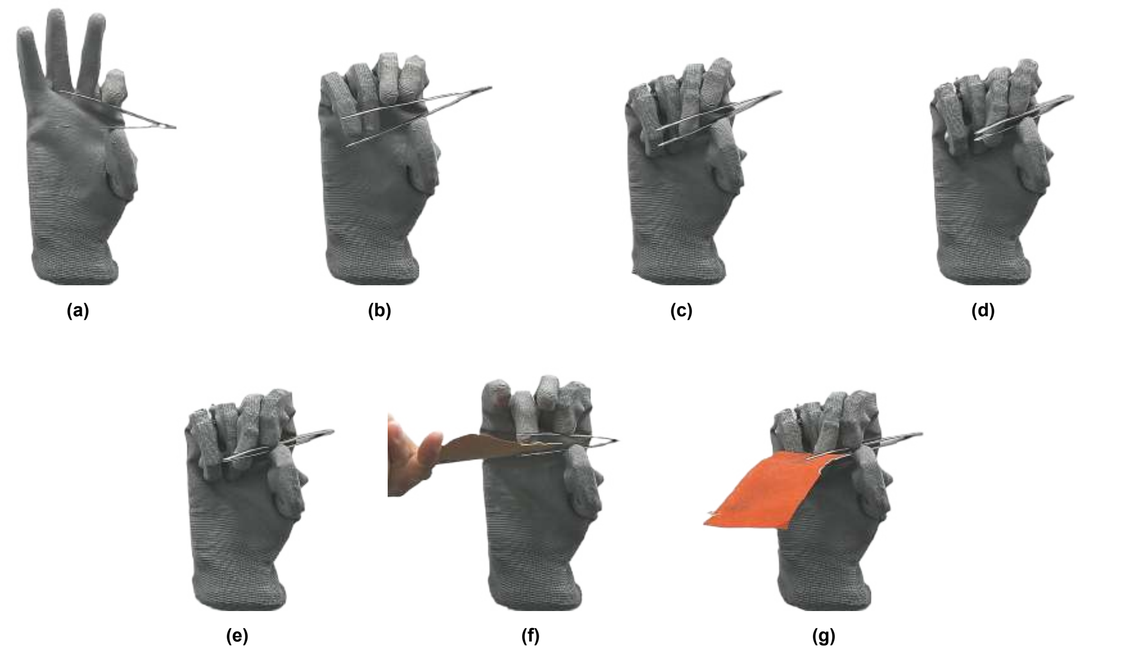


Figure 4.11: Photosequence of pinch task using a tweezer. An initial approach is made to hold the cherry independently, after which the hand is closed further to complete a fully closed pinch, which holds the cherry securely.

The next task involved the use of a surgical clamp. Once instigated, the clamp uses a ratcheting system to retain its clamping force. However, to engage the ratcheting system, a significant force is required, dependent on the size and rigidness of the object. In this case, the object to be grasped was an electrical wire, which was highly rigid and relatively thick. The surgical clamps 'ears' were placed over the index and thumb, after which the main actuation degree was used to close the hand sequentially. The first step brought the clamp in a more natural orientation, after which an electrical wire was held between the clamp's jaws. Additional actuation steps were then taken to close the hand further, totally securing the electrical wire in the clamp's jaws. The ratcheting system was not engaged for this object. However, it was able to withstand perturbations (pulling) on the wire and thus performed to satisfaction. Because the ratchet was not engaged, the fingers could also be extended back to the configuration seen in (a).

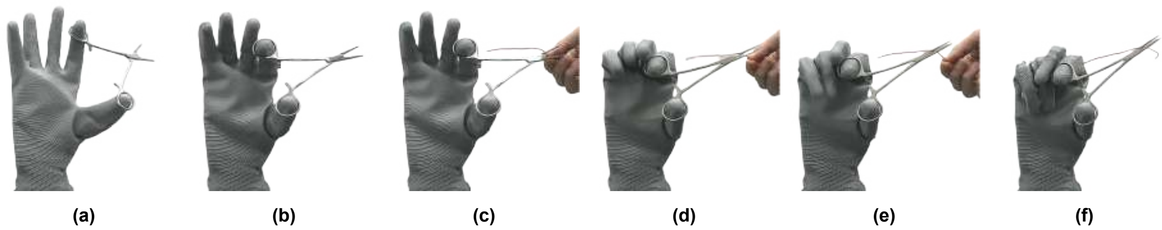


Figure 4.12: Photosequence of grasping task using a surgical clamp. An initial actuation command is sent to get the clamp in a realistic orientation, after which a wire is placed inside the jaw of the clamp. The hand is actuated further, closing the clamp further, thereby clamping the electric wire securely, completing the task.

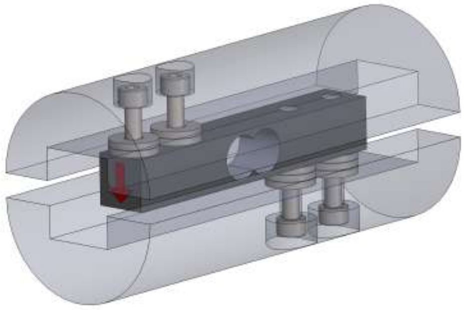
The final task to be performed was holding and activating a power drill. The power drill used in the task is a Bosch IXO V, weighing 300 g [30]. As seen in Figure 4.13, the hand could hold the drill independently in a natural way. The activation of the drill was more challenging, as in (c) the hand seemed to hold the drill well enough to activate it but did, in fact, not. In (d), however, the index finger gripped the activation trigger more successfully and was able to activate the power drill, thereby completing the task.

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

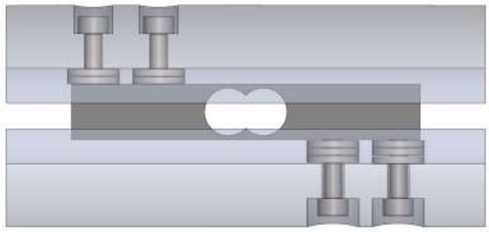
Figure 4.13: Power drill task where: (a) the drill is held in front of the hand, (b) the first actuation is done to hold the drill independently, (c) the actuation is at a maximum, but grasp did not result in the drill turning on, and (d) the actuation at a maximum which did turn on the power drill.

4.2.3. Grasp force measurements

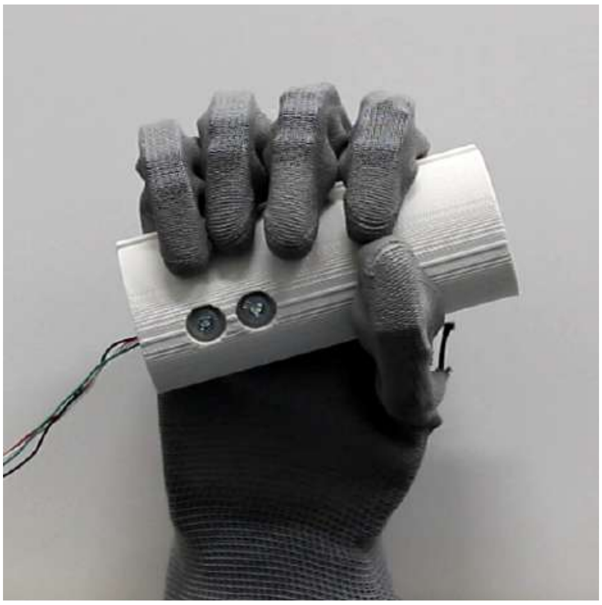
The grasping force was measured using the same load cell as the individual fingers. The load cell needs to be secured at one end and 'bent' at the other, so a custom cylinder part was made to create a graspable object. The CAD design for the cylinder can be seen in Figure 4.14, as well as the measurement setup. Two halves of a cylinder are 3D printed and connected to one end of the load cell using screws and spacers. By grasping the cylinder, the free space between the two cylinder halves allows the parts to move and create a bending moment on the load cell. The load cell is 1-dimensional, which means the measured force is only an indication of the true grasping force, as the pressure of the grasp comes from all directions. Nevertheless, the obtained measurements serve as an indication to compare the designed hand to prosthetic hands on the market, of which the results can be seen in Table 4.2. The same load cell measurement setup configuration was used as for the individual fingers: an HX711 load cell amplifier connected to a PC using a USB cable, with an Arduino script running and converting the measured data to output forces.



(a) Isometric view of load cell-cylinder design



(b) Side view of load cell-cylinder design



(c) Measurement setup

Figure 4.14: Design and measurement setup using the load cell-cylinder to measure grasp forces. The load cell is securely connected to the cylinder using screws and spacers.

Table 4.2: Force measurement comparison using the load cell-cylinder setup (* Max load cell measurement capability equal to 50 N)

	Prototype	I-Limb	Varispeed+
Measured forces [N]	36.35	40.29	50*

Note that the load cell used in the measurement setup was rated for maximum loads of 5 kg (50 N). Therefore, any measurement above this limit was capped at 50 N (the Varispeed+). Because the orientation of the cylinder during the grasp significantly impacts the measured results, this was kept as constant as possible, meaning that all the forces measured were in the direction from fingertips to palm (in-line with the screws in Figure 4.14).

4.2.4. Encoder data

The encoder data can serve as a visual representation of the movement of the hand after an actuation command. In Figure 4.15, the error (desired position - current measured position) is plotted over time during a free motion movement. A command of 270° is given as the desired position, and the motor can be seen to reduce this error smoothly and rapidly. There is no overshoot, and the steady-state error is around 10 degrees. This could be further reduced by tuning the PID parameters, but this was deemed sufficiently accurate during the prototype testing. The response time is fast at around 80 ms for an initial 270° error.

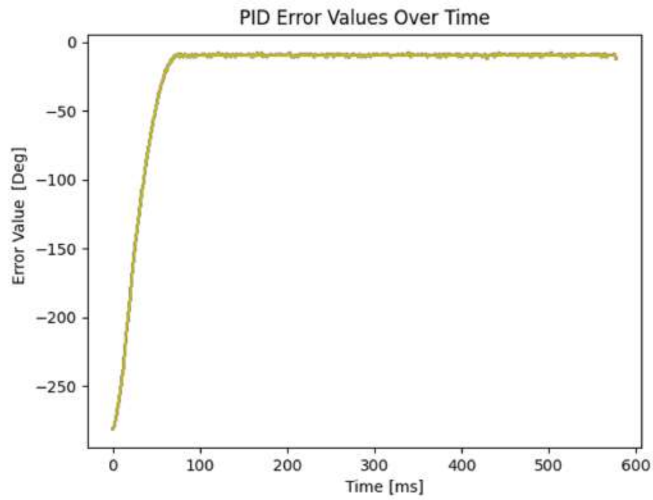


Figure 4.15: PID error values during a free movement grasp with positional command of 270°.

To investigate the behaviour of the hand further, the PID error values of a free movement were also compared to the grasping of an object. In Figure 4.16, the error value plot of a free movement with an initial error of 800° can be compared to the error value plot during the grasp of a plastic bottle. Note that the free movement again has a steady-state error of around 10 degrees, whereas the grasp of the plastic bottle results in a steady-state error of around 80 degrees. The larger error when grasping the object is likely due to the object resisting movement due to its high rigidity. Fine-tuning the PID parameters could reduce this but also affect performance in other situations. This highlights one of the challenges of finding the optimal balance in tuning the controller parameters. The response time of both plots is nearly identical, at around 1000ms. However, a difference in the error decrease phase can be seen throughout both movements, where the plastic bottle offers resistance and causes a decrease in the slope of the error reduction, which the controller then compensates.

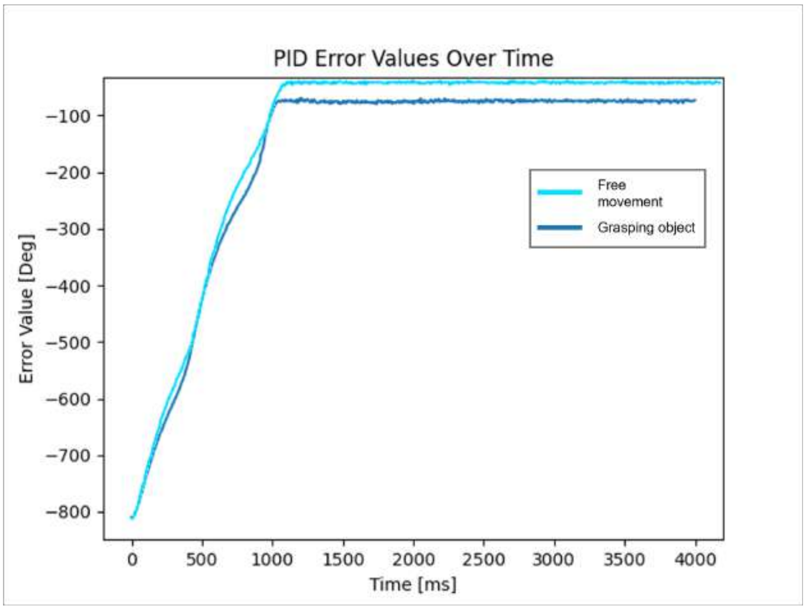


Figure 4.16: Comparison of PID error values between free movement and grasping of a plastic bottle. The plot illustrates the difference in steady-state error and error reduction behaviour between the two movements, highlighting the challenges of controller tuning in varied manipulation tasks.

4.3. Experimental inputs

This section provides an overview of the input parameters utilized throughout the experiments. The following Table 4.3 summarizes the key inputs employed in the finger and hand experiments, highlighting the best-performing inputs derived from multiple trial variations.

Table 4.3

Experiment name	Main motor command	Index command	Thumb command
Basic full closure	1100	0	0
Only index	0	1000	0
Only Thumb	0	0	750
Two-finger pinch-open	0	800	900
Sequence pinch-open	0→400	800→1000	700→700
Pinch-close	1100	800	500
Sequence Pinch-close	600→1050	700→900	500→500
Power rigid ball	900	0	0
Power soft ball	1050	0	0
Power Pear	900	0	0
Power Cola bottle	900	0	0
Pinch-Close Marker	550→700	700→700	350→350
Pinch-Close Cherries	550→900	450→900	450→450
Pinch-Open Disk	350	900	400
Pinch-Open Grapes	250	800	400
Pinch-Close Pen	550→700	700→700	350→350
Task Tweezer	0→400→800↔1150	800→1100	500→500
Task Surgical clamp	400→850	0	0
Task Power drill	500→ 1000	0	0
Force measurement: Individual fingers	1000	0	0
Force measurement: Grip force	1100	850	450

5 Discussion

This chapter draws conclusions and reflects on the performance of the designed prototype, improvement points and limitations, experimental suitability and results evaluation, and future work and alternative approaches.

In this thesis, we have developed an adaptive synergy actuated hand with additional parallel actuators to help increase functionality and performance. Experiments have demonstrated that the design performs promisingly and achieves all of its set design goals. The hand is highly functional, low-complexity, robust, and adequately anthropomorphic. The hand's high functional capabilities are shown by its successful completion of numerous grasping tasks and force comparisons to other prosthetic hands. The control of the hand is very straightforward, using low-complexity algorithms and simple fabrication techniques. A highlight of the robustness is the dislocatability of the fingers and their ability to perform adaptive grasps without the need for complex sensorization. Finally, the design is very comparable to that of a human hand in terms of size and finger build, except for some details, which will be explained further in a later section.

One of the main goals of this research was to further explore the possibilities within synergistic actuation design principles. Because proper multi-synergistic approaches take up significant amounts of space, reducing its viability for prosthetic hand design, an adaptation to this concept was sought that utilizes the synergistic approach but is more easily implemented in prosthetic hand design. Therefore, only the first synergy is used for the hand's main grasping functionality in the form of an adaptive synergistic design, with the addition of two small motors that enable a secondary actuation in the form of a pinch movement and an additional benefit of moving the thumb and index individually. Because the small motors each only actuate a single finger, the transmission architecture is a relatively easy and small incorporation into the hand. As seen in the experiments and results, the prototype works as intended, where the first synergy maximizes its adaptable underactuated nature to grasp objects of all shapes and sizes with ease, and the secondary actuators are utilized for objects that require a more delicate pinch. The versatility of the hand is significantly increased by the addition of these secondary actuators, especially when the actuation degrees are combined sequentially or in parallel. By changing the command inputs, the user can create various unique grasping patterns to fit the required needs optimally. Examples are additional index-thumb force during power grasp, open pinch, closed pinch, and intermediary forms of a closed pinch, where the remaining digits moved only partly. Additionally, the robustness of the dislocatable joints is shown, as they are able to return from dislocations and smaller perturbations. In conclusion, this novel actuation architecture, in combination with other design features, has proven to be a noteworthy improvement in increasing the functionality of low-complexity prosthetic hands.

Throughout the research of this thesis, several specific areas for improvement have emerged. Many of these improvements are already in place due to the iterative design process; however, some areas require additional attention. The primary issue during the prototype testing phase was the need for more performance of the smaller individual motors. The Pololu motors were readily available in the lab and were deemed good enough after some basic testing for the prototype. Unfortunately, issues began appearing after the complete incorporation of the motors, two of which were critical: the motor power and the encoder setup. First, the motor power, although initial testing showed the motors could move the fingers, even with return elastics installed, no further load tests were performed, which turned out to be one of the primary deficits, as the motors could not always overcome the higher power requirements needed for interaction with objects, such as pinching a thin object using only the small motors. The second critical issue was the flawed motor-encoder configuration used for these motors. The encoders are soldered to the motor's power pins, but these pins are too short to create a robust connection and are thus very susceptible to perturbations. Pololu has since tackled this issue, resulting in a new design pair for the motor and encoder, but unfortunately, it could not be incorporated into this project. This faulty connection is likely to have also impacted the power output performance of the motors, as this meant the motors were not always properly connected to their power source.

Another motor-related improvement was revealed by testing the individual finger forces of the hand. As seen in Table 4.1, the fingertip forces decrease from the little finger to the index finger. This is due to the friction losses and designed transmission architecture. The main tendon running from the thumb, through the two-digit loops, to the Maxon motor could be adapted to reverse this. The digits with preferred maximum fingertip forces are the index and middle fingers, but because the main motor is placed on the right side of the palm (dorsal view), it is almost directly in line with the little finger. Friction losses are thus the lowest for this finger, which could be altered by changing the routing to end with the index-middle finger loop, resulting in maximum fingertip force for this pair.

At the beginning of the project, the decision was made to design all four digits identically. The prime reason for that was to save time on design and production, and additionally, the kinematic behaviour of all digits would be identical. These assumptions were correct, although one unforeseen disadvantage surfaced when the glove was assimilated into the prototype. The digit and the individual phalanges were dimensioned to recreate human averages, which works well for the index-, middle- and ring fingers. However, the little finger has a significant size discrepancy compared to the other digits. This oversized little finger resulted in the need for relief cuts in the glove because the little finger was too long, creating tension throughout the glove, which impacted the hand's kinematic behaviour. The relief cuts functioned well; thus, the issue was mitigated sufficiently. However, two other solutions would be to fabricate a custom glove or, more importantly, change the dimensions of the little finger to a more realistic sizing. Because the little finger uses the same parts as the middle- and ring fingers to reduce fabrication time, they are the same size; however, to increase anthropomorphism, the little finger should be made smaller.

Various qualitative and quantitative experiments were conducted on the prototype to validate its performance and allow for comparison to other prosthetic hands on the market. The quantitative experiments in the form of force measurements primarily compare individual fingers or other prosthetic hands on the market. The individual fingertip force measurements merely compare the fingers and do not fully comprehend the actual fingertip forces. Since the hand is underactuated and thus adaptive to its environment, the digits that do not come into contact with the load cell continue moving to flex completely. In contrast, the fingertip to be measured is stopped due to the contact, where it applies a portion of its maximum force, which only increases when the other fingers are flexed completely. The inherent adaptability of the hand thus makes direct fingertip force measurements difficult. However, possible hand improvements were found because of these measurements, in the form of (un-) desirable behaviour with more force output in the less critical digit pair.

The load cell cylinder setup to measure the grasping force of the hand is not truly representative of the grasping force of the hand or the hands on the market. As mentioned, the load cell is one-dimensional and thus only measures force in one direction. Grasping force is more accurately represented as a two—or even three-dimensional pressure surrounding the grasped object. By measuring only one direction of the applied force, the measured forces are significantly undervalued compared to the true capabilities of the hands.

The qualitative experiments conducted in the form of grasping functional objects and performing tasks represent the hand's capabilities relatively well. In literature, a hand's performance is often related to the speed and accuracy during a grasping sequence of multiple objects, such as the Box and Block test [31], the Action Research Arm test [32], and the Virtual Egg test [33]. However, the current state of the hand does not allow for the movement of the hand in such a fashion. Instead, the hand remained stationary, allowing for the assessment of grasping capabilities alone. When the control architecture and moveability of the hand are further improved, for example, by using triggers or sensors, these qualitative experiments can be used as a good indication of quantified performance.

Overall, the obtained results merit further research. The promising results obtained with this prototype can be easily improved upon by implementing several changes to the design, including an upgrade to the motor, an optional internal transmission change to further benefit the primary digit pair, and a size reduction of the little finger to fit the surrounding glove better. By implementing these relatively simple changes in the design and making the design fully self-contained, further quantification of the hand's performance compared to others can validate its true benefits.

Finally, a note on future work concerns an adaptation to the actuation architecture. Currently, the individual actuation degrees used for the index and thumb are designed in parallel with the primary actuation degree. An interesting adaptation to this would be to change from a fully parallel structure to one both in series and in parallel. For example, a tendon runs from the thumb's motor through the thumb to the main motor, and the tendon from the index motor runs through the index finger to the fingertip of the middle finger. This actuation scheme creates an alternative similar to the augmented adaptive synergies in the Softhand 2 [25], which exploits the internal friction encountered in tendon systems and turns it into an advantage. By doing so, the Softhand 2 is able to execute in-hand manipulation, which is something this prototype was not able to do adequately and would thus serve as a valuable contribution. By varying combinations of parallel and in-series motors, new interesting combinations with unique, versatile capabilities can be researched, moving the field of prosthetic hand design forward, one grasp at a time.

References

[1] Francesca Cordella et al. “Literature review on needs of upper limb prosthesis users”. In: *Frontiers in neuroscience* 10 (2016), p. 209.

[2] J Merrilees. “Activities of daily living”. In: *Encyclopedia of the Neurological Sciences* 2 (2014), pp. 47–48.

[3] Timothy R. Dillingham and Diane W. Braza. *Chapter 108 - Upper Limb Amputations*. 2nd ed. W.B. Saunders, 2008, pp. 595–598. URL: `https://www.sciencedirect.com/science/article/pii/B9781416040071501103`.

[4] Diane W Braza and JN Yacub Martin. “Upper limb amputations”. In: *Essentials of physical medicine and rehabilitation* (2020), pp. 651–657.

[5] Lauren Trent et al. “A narrative review: current upper limb prosthetic options and design”. In: *Disability and Rehabilitation: Assistive Technology* 15.6 (Apr. 2019), pp. 604–613. DOI: `https://doi.org/10.1080/17483107.2019.1594403`.

[6] Charles E Clauser, John T McConville, and John W Young. *Weight, volume, and center of mass of segments of the human body*. Tech. rep. Antioch Coll Yellow Springs OH, 1969.

[7] Giulia Franchi et al. “The Baxter Easyhand: A robot hand that costs $150 US in parts”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (2015), pp. 2917–2922.

[8] Lianjun Wu et al. “Compact and low-cost humanoid hand powered by nylon artificial muscles”. In: *Bioinspiration & biomimetics* 12.2 (2017), p. 026004.

[9] TouchBionics. *i-limb quantum — precision, power, and intelligent motion*. 2023. URL: `https://www.ossur.com/nl-nl/prosthetics/armen/i-limb-quantum`.

[10] Taskaprosthetics. *Taska prosthetic hand*. 2022. URL: `https://www.taskaprosthetics.com/`.

[11] Ottobock. *michelangelo hand, the michelangelo hand helps you regain extensive freedom*. 2022. URL: `https://www.ottobock.com/en-us/product/8E500`.

[12] Clemens Eppner et al. “Exploitation of environmental constraints in human and robotic grasping”. In: *The International Journal of Robotics Research* 34.7 (2015), pp. 1021–1038.

[13] C Piazza et al. “A century of robotic hands”. In: *Annual Review of Control, Robotics, and Autonomous Systems* 2 (2019), pp. 1–32.

[14] Jr. Benny M. Hillberry Allen S. Hall. “Rolling contact joint”. 3932045A. 1976. URL: `https://patents.google.com/patent/US3932045A`.

[15] Jesse R Cannon, Craig P Lusk, and Larry L Howell. “Compliant rolling-contact element mechanisms”. In: *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*. Vol. 47446. 2005, pp. 3–13.

[16] C. Piazza et al. “The SoftHand Pro-H: A Hybrid Body-Controlled, Electrically Powered Hand Prosthesis for Daily Living and Working”. In: *IEEE Robotics & Automation Magazine* 24.4 (2017), pp. 87–101. DOI: `10.1109/MRA.2017.2751662`.

[17] Huan Liu et al. “The mero hand: A mechanically robust anthropomorphic prosthetic hand using novel compliant rolling contact joint”. In: *2019 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM)* (2019), pp. 126–132.

[18] Annick Mottard, Thierry Laliberté, and Clément Gosselin. “Underactuated tendon-driven robotic/prosthetic hands: Design issues.” In: *Robotics: Science and Systems* 7 (2017).

[19] Marco Santello, Martha Flanders, and John F Soechting. “Postural hand synergies for tool use”. In: *Journal of neuroscience* 18.23 (1998), pp. 10105–10115.

[20] Jens Vertongen et al. “Mechanical aspects of robot hands, active hand orthoses, and prostheses: A comparative review”. In: *IEEE/ASME Transactions on Mechatronics* 26.2 (2020), pp. 955–965.

[21] Danilo Estay et al. “Development and implementation of an anthropomorphic underactuated prosthesis with adaptive grip”. In: *Machines* 9.10 (2021), p. 209.

[22] Manuel G Catalano et al. “Adaptive synergies for the design and control of the Pisa/IIT SoftHand”. In: *The International Journal of Robotics Research* 33.5 (2014), pp. 768–782.

[23] Marco Gabiccini et al. “On the role of hand synergies in the optimal choice of grasping forces”. In: *Autonomous Robots* 31 (2011), pp. 235–252.

[24] Giorgio Grioli et al. “Adaptive synergies: An approach to the design of under-actuated robotic hands”. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems* (2012), pp. 1251–1256. DOI: 10.1109/IROS.2012.6385881.

[25] Cosimo Della Santina et al. “Toward dexterous manipulation with augmented adaptive synergies: The pisa/iit softhand 2”. In: *IEEE Transactions on Robotics* 34.5 (2018), pp. 1141–1156.

[26] Grog LLC. *Whoopie sling*. URL: https://www.animatedknots.com/whoopie-sling-knot.

[27] Alexander Buryanov and Viktor Kotiuk. “Proportions of hand segments”. In: *Int. J. Morphol* (2010), pp. 755–758.

[28] *Pololu DRV8833 dual motor driver*. URL: https://www.pololu.com/product/2130.

[29] Scott Bryson. “Application Brief: Absolute Angle Measurements for Rotational Motion Using Hall-Effect Sensors”. In: *Texas Instruments* (2023), pp. 1–4.

[30] *IXO 5 accuboormachine*. URL: https://www.bosch-diy.com/nl/nl/p/ixo-5-06039a8000.

[31] Virgil Mathiowetz et al. “Adult norms for the Box and Block Test of manual dexterity”. In: *The American journal of occupational therapy* 39.6 (1985), pp. 386–391.

[32] Nuray Yozbatiran, Lucy Der-Yeghiaian, and Steven C Cramer. “A standardized approach to performing the action research arm test”. In: *Neurorehabilitation and neural repair* 22.1 (2008), pp. 78–90.

[33] Francesco Clemente et al. “Non-Invasive, Temporally Discrete Feedback of Object Contact and Release Improves Grasp Control of Closed-Loop Myoelectric Transradial Prostheses”. In: *IEEE Transactions on Neural Systems and Rehabilitation Engineering* 24.12 (2016), pp. 1314–1322. DOI: 10.1109/TNSRE.2015.2500586.

A

Source Code

Python source code used on the command console. The code creates a GUI element that can communicate with the Serialsocket on which the microcontroller is connected.

```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtGui import QFont
import numpy as np
import pyqtgraph as pg
import time
import signal
import sys
import collections
from Worker import Worker
from UDPtest import SerialSocket
import matplotlib.pyplot as plt

DEFAULT_STYLE = """
QProgressBar{
    border: 2px solid grey;
    border-radius: 5px;
    text-align: center
}

QProgressBar::chunk {
    background-color: green;
    width: 10px;
    margin: 1px;
}
"""
DEFAULT_STYLE_1 = """
QProgressBar{
    border: 2px solid grey;
    border-radius: 5px;
    text-align: center
}

QProgressBar::chunk {
    background-color: lightblue;
    width: 10px;
    margin: 1px;
}
"""

class SerialSocketMock:
    def __init__(self):
        self.mock_data = iter([b'Status:␣Mock␣data␣received\n'])

    def send_data(self, data):
        print("Mock:␣Sending␣data:", data)
```

```python

    def receive_data(self):
        try:
            return next(self.mock_data)
        except StopIteration:
            return b''

class App(QMainWindow):
    def __init__(self, parent=None):
        super(App, self).__init__(parent)

        self.setWindowTitle("GUI␣Robotic␣Hand␣Control␣panel")

        self.esp32 = SerialSocketMock()
        self.startTime = time.time()

        # Create QLineEdit widgets for hand, index, and thumb input
        self.hand_input = QLineEdit(self)
        self.index_input = QLineEdit(self)
        self.thumb_input = QLineEdit(self)

        # Set up the main layout with QGridLayout
        main_layout = QGridLayout()
        main_layout.setContentsMargins(1, 1, 1, 1)  # Set small margins
        main_layout.setSpacing(1)  # Set small spacing

        for i in range(3):
            main_layout.setColumnStretch(i, 1)

        # Create button widgets and add them to specific positions in the layout
        savePosition = QPushButton('Start␣ESP32', self)
        savePosition.clicked.connect(self.start)
        main_layout.addWidget(savePosition, 0, 0, 1, 1)  # (row, column, rowspan, colspan)

        readHandPosition = QPushButton('Read␣Hand␣Position', self)
        readHandPosition.clicked.connect(self.readPosition)
        main_layout.addWidget(readHandPosition, 0, 1, 1, 1)

        stop_ = QPushButton('Stop', self)
        stop_.clicked.connect(self.stopHand)
        main_layout.addWidget(stop_, 0, 2, 1, 1)



        # Add QLabel as the title for each input
        hand_title = QLabel("Hand<br>Input:␣degrees<br>(+Close␣/␣-Open)")
        index_title = QLabel("Index<br>Input:␣encoder␣Ticks<br>(+Close␣/␣-Open)")
        thumb_title = QLabel("Thumb<br>Input:␣encoder␣Ticks<br>(+Close␣/␣-Open)")

        title_font = QFont("Arial", 12, QFont.Bold)
        hand_title.setFont(title_font)
        index_title.setFont(title_font)
        thumb_title.setFont(title_font)

        main_layout.addWidget(hand_title, 1, 0, 1, 1)
        main_layout.addWidget(self.hand_input, 2, 0, 1, 1)
        # send_hand_button = QPushButton('Send Hand', self)
        # send_hand_button.clicked.connect(self.send_hand)
        # main_layout.addWidget(send_hand_button, 3, 0, 1, 1)

        main_layout.addWidget(index_title, 1, 1, 1, 1)
        main_layout.addWidget(self.index_input, 2, 1, 1, 1)
        # send_index_button = QPushButton('Send Index', self)
        # send_index_button.clicked.connect(self.send_index)
        # main_layout.addWidget(send_index_button, 3, 1, 1, 1)

        main_layout.addWidget(thumb_title, 1, 2, 1, 1)
        main_layout.addWidget(self.thumb_input, 2, 2, 1, 1)
        # send_thumb_button = QPushButton('Send Thumb', self)
        # send_thumb_button.clicked.connect(self.send_thumb)
        # main_layout.addWidget(send_thumb_button, 3, 2, 1, 1)
```

```

        # send_hand_button = QPushButton('Send Hand Value', self)
        # send_hand_button.clicked.connect(lambda: self.send_value(self.hand_input, self.
            moveHandCCW))
        # main_layout.addWidget(send_hand_button, 3, 0, 1, 1)

        # send_index_button = QPushButton('Send Index Value', self)
        # send_index_button.clicked.connect(lambda: self.send_value(self.index_input, self.
            moveIndex))
        # main_layout.addWidget(send_index_button, 3, 1, 1, 1)

        # send_thumb_button = QPushButton('Send Thumb Value', self)
        # send_thumb_button.clicked.connect(lambda: self.send_value(self.thumb_input, self.
            moveThumb))
        # main_layout.addWidget(send_thumb_button, 3, 2, 1, 1)


        # send_hand_button = QPushButton('Send Hand Value', self)
        # send_hand_button.clicked.connect(lambda: self.send_value(self.hand_input, self.
            moveHandCCW))
        # main_layout.addWidget(send_hand_button, 3, 0, 1, 1)

        # send_index_button = QPushButton('Send Index Value', self)
        # send_index_button.clicked.connect(lambda: self.send_value(self.index_input, self.
            moveIndex))
        # main_layout.addWidget(send_index_button, 3, 1, 1, 1)

        # send_thumb_button = QPushButton('Send Thumb Value', self)
        # send_thumb_button.clicked.connect(lambda: self.send_value(self.thumb_input, self.
            moveThumb))
        # main_layout.addWidget(send_thumb_button, 3, 2, 1, 1)
        send_values_button = QPushButton('Send␣Values', self)
        send_values_button.clicked.connect(self.send_values)
        main_layout.addWidget(send_values_button, 3, 0, 1, 3)

        central_widget = QWidget()
        central_widget.setLayout(main_layout)
        self.setCentralWidget(central_widget)

        self.angle = 0
        self.curr = collections.deque(maxlen=100)
        self.que = collections.deque(maxlen=100)
        for i in range(100):
            self.que.append(0)
        self.pos = collections.deque(maxlen=100)
        for i in range(100):
            self.pos.append(0)

        self.plotRunning = True
        self.plotting = True

        self.threadpool = QThreadPool()
        self.readData = Worker(self.print_esp32)
        self.readData.signals.result.connect(self.readData.print_output)
        self.readData.signals.progress.connect(self.readData.progress_fn)
        self.threadpool.start(self.readData)

        self.t = np.linspace(0, 100, 100)

        self.status_message = bytes([0])
        self.ESP32connected = False

    def updatePlots(self):
        self.ErrorPlot.clear()
        self.ErrorPlot.plot(self.t, self.que)
        self.PositionPlot.clear()
        self.PositionPlot.plot(self.t, self.pos)
        self.statusBox.setPlainText(self.status_message.decode())

    def print_esp32(self, progress_callback):
        while self.plotting:
```

```
            d = self.esp32.receive_data()
            if d[0:5] == b'Error':
                self.que.append(float(d[7:12].decode()))
            elif d[0:5] == b'RAngle':
                self.angle.append(float(d[7:12].decode()))
            elif d[0:8] == b'Position':
                if len(d) > 15:
                    if d[10] == b'-':
                        self.pos.append(-float(d[11:15].decode()))
                    else:
                        self.pos.append(float(d[11:15].decode()))
            elif d[0:5] == b'Speed':
                self.vel.append(float(d[7:12].decode()))
            elif d[0:5] == b'Current':
                self.curr.append(float(d[7:12].decode()))
            elif d[0:5] != b'':
                print(d.decode())
            time.sleep(0.005)


    # self.esp32.send_data: input a list with following variables: [cmd, targethand,
        targetIndex, targetThumb]
    def start(self):
        self.esp32.send_data([4, 0, 0, 0])

    def stopHand(self):
        self.esp32.send_data([1, 0, 0, 0])

    def moveHandCCW(self, degrees):
        degrees = -degrees
        self.esp32.send_data([2, degrees,0, 0])

    def moveThumb(self, ticks_thumb):
        ticks = ticks_thumb
        time.sleep(.002)
        self.esp32.send_data([5, 0, 0, ticks])

    def moveIndex(self, ticks):
        ticks = -ticks
        self.esp32.send_data([6, 0, ticks, 0])

    def movePinch(self, tick_index, tick_thumb):
        tick_index = -tick_index
        tick_thumb = tick_thumb
        self.esp32.send_data([7, 0, tick_index, tick_thumb])

    def moveHandAndFingers(self, degrees, tick_index, tick_thumb):
        degrees = -degrees
        tick_index = -tick_index
        tick_thumb = tick_thumb
        self.esp32.send_data([8,degrees,tick_index,tick_thumb])

    def readPosition(self):
        self.esp32.send_data([3, 0, 0, 0])

    def writePosition(self):
        self.esp32.send_data([6, int(self.positionDegree.text())])

    def closeEvent(self, event):
        self.plotRunning = False
        self.plotting = False
        self.esp32.send_data([4, 0, 0, 0])
        print("Closing")



    def send_values(self):
        try:
            hand_value = self.hand_input.text()
            index_value = self.index_input.text()
            thumb_value = self.thumb_input.text()
```

```
            if hand_value and index_value and thumb_value:
                try:
                    hand_value = float(hand_value)
                    index_value = float(index_value)
                    thumb_value = float(thumb_value)
                    # print(index_value, thumb_value)
                    self.moveHandAndFingers(hand_value, index_value, thumb_value)
                except ValueError:
                    print("Invalid␣input.␣Please␣enter␣valid␣numeric␣values.")
            else:
                print("Invalid␣input.␣Please␣enter␣valid␣numeric␣values␣in␣all␣text␣boxes.")
            # if index_value and thumb_value:
            #     try:
            #         index_value = float(index_value)
            #         thumb_value = float(thumb_value)
            #         print(index_value, thumb_value)
            #         self.movePinch(index_value, thumb_value)
            #     except ValueError:
            #         print("Invalid input. Please enter valid numeric values.")
            # else:
            #     # Run different code when either index_value or thumb_value is not None
            #     if index_value:
            #     # Code for when only index_value is not None
            #         index_value = float(index_value)
            #         self.moveIndex(index_value)

            # if thumb_value:
            #     # Code for when only thumb_value is not None
            #         thumb_value = float(thumb_value)
            #         self.moveThumb(thumb_value)
            # if hand_value:
            #     hand_value = float(hand_value)
            #     self.moveHandCCW(hand_value)


        except ValueError:
            print("Invalid/No␣input.␣Please␣enter␣valid␣numeric␣values.")


if __name__ == '__main__':
    app = QApplication(sys.argv)
    thisapp = App()
    thisapp.resize(1200, 200)
    thisapp.show()
    try:
        def signal_handler(signal, frame):
            thisapp.plotRunning = False
            thisapp.plotting = False
            print('You␣pressed␣Ctrl+C!')
            sys.exit(0)

        signal.signal(signal.SIGINT, signal_handler)
        signal.signal(signal.SIGTERM, signal_handler)
        signal.signal(signal.SIGILL, signal_handler)
        signal.signal(signal.SIGABRT, signal_handler)
        app.exec_()
    except Exception as e:
        print('Error:␣' + str(e))
        thisapp.plotting = False
        thisapp.plotRunning = False
        sys.exit(0)
```

C++ source code that is run on the microcontroller. The primary function is the task manager and running PID algorithms and data handling.

```
#include "myo.h"
#include "c3dhall11.h"
#include "PID.h"
#define MIN_MAX_MAGNET 10
```

```
#define pi 3.14
#define cw 0 // close
#define ccw 1 //open

armband myo;                // Myo BLE Armband

uint8_t *emgData_raw = NULL;
unsigned emgData[2] ={0};// {0,3,4,8};

static unsigned emgCnt = 0;
static unsigned status_ = 0;
static uint8_t Q = 0;
static uint8_t oldQ = 5;
static const int EMGtaskCore = 0;
static const int MOTORtaskCore = 1;
static float totalAngle;
static int numberOfTurns = 0;
static const int SOB_pin = 2;
static int encoderOffset = 0;
static int OldPos[4] = {0};
static float speed = 0;
static uint32_t oldTime = millis();
static bool initialized = false;
static float x_adjustment = -4.2;//20.14;
static float y_adjustment = -4.75;//30.00;

// PID values Joost
int counter = 0;
int pos = 0;
long prevT =0;
float eprev = 0;
float eprev_hand = 0;
float eprev_thumb = 0;
float eprev_index = 0;
float eintegral=0;
float Kp_main= 0.8f;
float Ki_main= 0.f;
float Kd_main= 0.02f;

float Kp_hand= 0.8f;
float Ki_hand= 0.f;
float Kd_hand= 0.02f;

float Kp_index= 1.8f; //Begon op .8f
float Ki_index= 0.f;
float Kd_index= 0.08f;

float Kp_thumb= 2.5f; //Begon op .8f
float Ki_thumb= 0.f;
float Kd_thumb= 0.08f;

const char* names[3] = {"Rest","Flex", "Extend"};
bool emgalert = false;

int directionPin = 6;
static int emg_rms = 0;
const int mainMotorPin1 = 26; //!!
const int mainMotorPin2 = 25; //!!
const int thumbMotorPin1 = 12;
const int thumbMotorPin2 = 17;
const int indexMotorPin1 = 18; //!!
const int indexMotorPin2 = 13; //!!
const int motorChannel[6] = {0,1,2,3,4,5};

int motorCurrent = 0;

const int EncoderThumbA = 4;
const int EncoderThumbB = 19;
const int EncoderIndexA = 2; //!!
const int EncoderIndexB = 5; //!!

```

```

int last_state_IA = 0;
int last_state_TA = 0;

volatile long encoderThumbCount = 0;
volatile long encoderIndexCount = 0;

TwoWire I2Cone = TwoWire(0);
C3dhall11 encoder;
c3dhall11_data_t sensor_data;
bool deleteTask;

TaskHandle_t xHandle = NULL;

void initPosition(bool *delTask){
  int pos_temp = 0;
  encoderOffset = 0;
  for(uint8_t i = 0; i<50; i++){
      checkQuadrant(readAngle());
      // if (round(totalAngle) != round(113.25)){
      pos_temp += totalAngle;
      // Serial.println(numberOfTurns);
      // }
      // else{
      //   Serial.println("Angle is 113.25, if this is the actual average please move the motor slightly..");
      // }
      delay(20);
  }
  // Serial.print(pos_temp);
  encoderOffset = (pos_temp/49);
  Serial.print("EncoderOffset = ");Serial.println(encoderOffset);
  pos_temp=0;

  *delTask = true;
}

/* ****************************************************************************************************
    SET CALLBACKS WHEN RECEIVING DATA
 ****************************************************************************************************
 */

void batteryCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* pData, size_t length, bool isNotify) {
  myo.battery = pData[0];
  Serial.print("Battery: ");
  Serial.println(myo.battery);
}

void imuCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* pData, size_t length, bool isNotify) {
  Serial.print ("EMG: \t");
  for (int i = 0; i < length; i++) {
    Serial.print(pData[i]);
    Serial.print("\t");
  }
  Serial.println(millis());
}

void emgCallback(BLERemoteCharacteristic* pBLERemoteCharacteristic, uint8_t* pData, size_t length, bool isNotify) {
  emgData_raw = pData;
  /*Serial.print ("EMG: \t");
  for (int i = 0; i < length; i++) {
    Serial.print(pData[i]);
    Serial.print("\t");
  }
  Serial.println(millis());
  */
```

```
}
void readCurrent(void * pvParameters){
  while(true){
    Serial.println(analogRead(15)*3);
    delay(500);
  }
}
void processSerialData(void * pvParameters ){
  int targetPos_hand = 0;
  int targetPos_index = 0;
  int targetPos_thumb = 0;  // Added for the thumb target position
  float error = 0.0;
  int cmd = 0;
  char *token;
  char inputString[50];
  int input[8];
  // while(true) {
  //   if(int j = Serial.available()){
  //     for (uint8_t i = 0; i < 3; i++) {
  //       input[i] = Serial.parseInt();
  //     }
  //     for (uint8_t i = 3; i < j; i++) {
  //       Serial.read();
  //     }
  //     Serial.print("Command: "); Serial.println(input[0]);
  //     Serial.print("Target Position: "); Serial.println(input[1]);
  //     Serial.print("Target Position 2: "); Serial.println(input[2]);
  //     cmd = input[0];
  //     targetPosition = input[1];
  //     targetPos_thumb = input[2];

  //   }
  while (true) {
    if (int j = Serial.available()) {
      Serial.readBytesUntil('\n', inputString, sizeof(inputString)); // Read until newline
      token = strtok(inputString, ",");

      // Parse up to 3 integers
      for (uint8_t i = 0; i < 4 && token != NULL; i++) {
        input[i] = atoi(token);
        token = strtok(NULL, ",");
      }

      // Skip the rest of the input
      while (Serial.available()) {
        Serial.read();
      }

      Serial.print("Command:␣"); Serial.println(input[0]);
      Serial.print("Target␣Hand:␣"); Serial.println(input[1]);
      Serial.print("Target␣Index:␣"); Serial.println(input[2]);
      Serial.print("Target␣Thumb:␣"); Serial.println(input[3]);

      cmd = input[0];
      targetPos_hand = input[1];
      targetPos_index = input[2];
      targetPos_thumb = input[3];
    }

    if(cmd == 1){
      ledcWrite(cw, 0);
      ledcWrite(ccw, 0);
      ledcWrite(2,0);
      ledcWrite(3,0);
      ledcWrite(4,0);
      ledcWrite(5,0);
      cmd = 0;
    }
    else if(cmd == 2){
      pidHand(&targetPos_hand, &cmd);
        // pidPosition(&targetPosition, &error,&cmd);
```

```
    }
    else if(cmd == 3){
      checkQuadrant(readAngle());
      Serial.print("Hand␣total␣angle␣=␣");Serial.println(totalAngle);delay(2);
      Serial.print("Thumb␣encoder␣count␣=␣");Serial.println(encoderThumbCount);delay(2);
      Serial.print("Index␣encoder␣count␣=␣");Serial.println(encoderIndexCount);delay(2);
      cmd = 0;
    }
    else if(cmd==4){
      initialize();
      cmd = 0;
    }
    else if (cmd==5){
      // int c = encoderThumbCount;
      // int target = targetPosition + c;
      pidThumbMotor(&targetPos_thumb, &cmd);
    }
    else if (cmd==6){
      // int c = encoderIndexCount;
      // int target = targetPosition + c;
      pidIndexMotor(&targetPos_index, &cmd);
    }
    else if (cmd==7){
      pidPinch(&targetPos_index, &targetPos_thumb, &cmd);
    }
    else if (cmd==8){
      pidHandAndFingers(&targetPos_hand, &targetPos_index, &targetPos_thumb, &cmd);
    }
    else{
      if(initialized){
        ledcWrite(0, 0);
        ledcWrite(1, 0);
        ledcWrite(2, 0);
        ledcWrite(3, 0);
        ledcWrite(4, 0);
        ledcWrite(5, 0);

      }

    }
    delay(4);

  }
}

// void processEMGData(void * pvParameters ){
//   while(myo.connected) {
//     if( emgData_raw != NULL){
//       emgData[0] += (abs(int8_t(emgData_raw[3])) + abs(int8_t(emgData_raw[4])))/2;
//       emgData[1] += (abs(int8_t(emgData_raw[7])) + abs(int8_t(emgData_raw[0])))/2;
//       emgCnt++;

//       if(emgCnt > 4){
//         int8_t extensors = (emgData[0]/emgCnt);
//         int8_t flexors =  (emgData[1]/emgCnt);
//         if(flexors > threshold && extensors < (threshold/2)){
//           emg_rms = abs(flexors);
//           status_ = 1;
//         }
//         else if(extensors >threshold && flexors < (threshold/2)){
//           emg_rms = abs(extensors);
//           status_ = 2; //Serial.println("Wrist Extension.");
//         }
//         else
//           status_ = 0; //Serial.println("Rest");*/

//         emgCnt = 0;
//         emgData[0] = 0;
//         emgData[1] = 0;
//         emgData[2] = 0;
//         emgData[3] = 0;
```

```
//       }
//     }
//     delay(10);
//   }
// }

// void MyoDisconnectTask( void * pvParameters ){
//   while(true){
//     // Detect disconnection
//     if (!myo.connected) {
//       status_ = 0;
//       Serial.println ("Device disconnected: reconnecting...");
//       myo.connect();
//       Serial.println (" - Connected");
//       myo.set_myo_mode(myohw_emg_mode_send_emg,                // EMG mode
//                        myohw_imu_mode_none,                    // IMU mode
//                        myohw_classifier_mode_disabled);        // Classifier mode
//       myo.emg_notification(TURN_OFF)->registerForNotify(emgCallback);
//     }
//     delay(100);
//   }
// }

void readPositionTask(void *pvParameters){
  bool *delete_;
  delete_ = (bool *)pvParameters;
  while(true){
    checkQuadrant(readAngle());
    // Serial.print("Hand Position: ");
    // Serial.println(totalAngle);
    // Serial.print("Thumb Count: ");
    // Serial.println(encoderThumbCount);

    // Serial.print("measured_xy_angle: ");
    // Serial.println(sensor_data.angle);
    delay(1);
    if(*delete_ == true){
    if( xHandle != NULL )
    {
      vTaskDelete( xHandle );
    }
    }
  }
}

// void MotorControlTask(void *pvParameters){
//   while(true){
//     if(totalAngle>0 || totalAngle < 360){
//       switch(status_){
//         case 0:
//             ledcWrite(0, 0);
//             ledcWrite(1, 0);
//             Serial.print("Status: ");Serial.print(names[status_]); Serial.print("RMS: ");
    Serial.println(emg_rms);
//             break;
//         case 1:
//             //digitalWrite(directionPin, HIGH); // sets the digital pin 13 on
//             ledcWrite(0, 0);
//             ledcWrite(1, emg_rms);
//             Serial.print("Status: ");Serial.print(names[status_]); Serial.print("RMS: ");
    Serial.println(emg_rms);
//             break;
//         case 2:
//             //digitalWrite(directionPin, LOW); // sets the digital pin 13 on
//             ledcWrite(0, emg_rms);
//             ledcWrite(1, 0);
//             Serial.print("Status: ");Serial.print(names[status_]); Serial.print("RMS: ");
    Serial.println(emg_rms);
//             break;
//         default:
//             ledcWrite(0, 0);
```

```
//              ledcWrite(1, 0);
//              Serial.print("Status: ");Serial.print(names[status_]); Serial.print("RMS: ");
    Serial.println(emg_rms);
//              break;

//        }
//      }

//      delay (100);
//    }
// }

void checkQuadrant(float encoderPos)
{
  // ----- Quadrant 1
  if (encoderPos >= 0 && encoderPos <= 90)    Q = 1;
  else if (encoderPos > 90 && encoderPos <= 180) Q = 2;
  else if (encoderPos > 180 && encoderPos <= 270) Q = 3;
  else if (encoderPos > 270 && encoderPos < 360) Q = 4;

  if (Q != oldQ)                   // if we changed quadrant
  {
    if (Q == 1 && oldQ == 4){numberOfTurns++;}                                          //
        4 --> 1 transition: CW rotation
    else if (Q == 4 && oldQ == 1) {numberOfTurns--;}
                                                             // 1 --> 4 transition: CCW rotation
    oldQ = Q;                      //update to the current quadrant
  }
  totalAngle = (numberOfTurns * 360) + encoderPos - encoderOffset;            //number of
      turns (+/-) plus the actual angle within the 0-360 range
  // Serial.print("encoderPos = ");Serial.println(encoderPos);
  // Serial.print("NoT: ");Serial.print(numberOfTurns);Serial.print("ePos: ");Serial.print(
      encoderPos);Serial.print("eOff: ");Serial.println(encoderOffset);
}
float readAngle(){
  if (C3DHALL11_OK == encoder.read_data(&sensor_data)){

        float angle = ((atan2(sensor_data.x_axis+x_adjustment, sensor_data.y_axis+
            y_adjustment)+pi) * 180/pi);
        // Serial.print("RAngle: ");Serial.println(angle);
        return angle;
  }
  else return -0.f;
}

void updateEncoderThumb() {
  // Read the current state of both encoder pins
  // int stateA = digitalRead(EncoderThumbA);
  // int stateB = digitalRead(EncoderThumbB);

  // // Combine the states to get the quadrature encoding
  // int encoderValue = (stateA << 1) | stateB;

  // // Update the encoder count based on the quadrature encoding
  // switch (encoderValue) {
  //   case 0: case 3:
  //     encoderThumbCount++;  // Clockwise rotation
  //     break;
  //   case 1: case 2:
  //     encoderThumbCount--;  // Counterclockwise rotation
  //     break;
  //   default:
  //     // Invalid state, do nothing
  //     break;
  // }
  int state_TA = digitalRead(EncoderThumbA);
  int state_TB= digitalRead(EncoderThumbB);


  if (state_TA != last_state_TA){
              if (state_TA == state_TB){
```

```
                    //Clockwise rotation
                    encoderThumbCount ++;
                }
                else{
                    //Counterclockwise rotation
                    encoderThumbCount --;
                }
  }

          //Update last state
          last_state_TA = state_TA;

          //Optional: You may want to add a delay here to avoid rapid position changes
          // delay(1);

}

void updateEncoderIndex() {
  // Read the current state of both encoder pins
//   int stateC = digitalRead(EncoderIndexA);
//   int stateD = digitalRead(EncoderIndexB);

//   // Combine the states to get the quadrature encoding
//   int encoderValue2 = (stateC << 1) | stateD;

//   // Update the encoder count based on the quadrature encoding
//   switch (encoderValue2) {
//     case 0: case 3:
//       encoderIndexCount++;  // Clockwise rotation
//       break;
//     case 1: case 2:
//       encoderIndexCount--;  // Counterclockwise rotation
//       break;
//     default:
//       // Invalid state, do nothing
//       break;
//   }
// }
int state_IA = digitalRead(EncoderIndexA);
int state_IB= digitalRead(EncoderIndexB);


if (state_IA != last_state_IA){
            if (state_IA == state_IB){
                //Clockwise rotation
                encoderIndexCount ++;
            }
            else{
                //Counterclockwise rotation
                encoderIndexCount --;
            }
}

        //Update last state
        last_state_IA = state_IA;

        //Optional: You may want to add a delay here to avoid rapid position changes
        // delay(1);
}

// int connect_Myo(){
//   Serial.println ("Connecting to Myo Armband");
//   myo.debug = true;
//   myo.connect();                                    // Connect to the myo
//   Serial.println (" - Connected");

//   myo.set_myo_mode_();
//   delay(10);
//   Serial.println ("EMG mode set!");
//   myo.set_sleep_mode(1);
//   Serial.println ("Sleep mode set!");
```

```

//   //myo.battery_notification(TURN_ON)->registerForNotify(batteryCallback);
//   //myo.gesture_notification(TURN_OFF)->registerForNotify(gestureCallback);
//   myo.registerEMGCallback(emgCallback);
//   Serial.println ("EMG set!");

//   delay(500);
//   return 0;

// }

void pidHand(int* setpoint, int *status){
// int myNum[3] = {10, 20, 30};
  // int motorChannel[2] = {0,1};
  checkQuadrant(readAngle()); // returns totalAngle
  int e = *setpoint-totalAngle;
  // Serial.print("e = ");
  // Serial.println(e);

  // if (totalAngle<-720){
  //   *status = 0;
  //   return;
  // }

  // if ((e > 8) || (e < -8)){

  long currT = micros();

  float deltaT = ((float)(currT-prevT))/1.0e6;
  prevT = currT;

  // int e = *setpoint-totalAngle;

  float dedt = (e-eprev)/(deltaT);

  eintegral = eintegral + e*deltaT;

  // Control signal U
  float u = Kp_main*e + Kd_main*dedt + Ki_main*eintegral;

  // motor power
  float pwr = fabs(u);
  if (pwr>180){
    pwr=180;
  }
  Serial.println(e);

  // if (pwr < 30){
  //   pwr = 30;
  // }

  int dir = 1;
  if (u<0){
    dir = -1;
  }
  // Serial.print("pwr set at: ");
  // Serial.println(pwr);
  int moto1 = motorChannel[0];
  int moto2 = motorChannel[1];
  setMotor(moto1, moto2, dir,pwr);
  // Serial.print("Error e: ");
  // char errorprint = e;
  // Serial.println(errorprint);

  eprev = e;
    // checkQuadrant(readAngle());


  // if ((e < 5) && (e > -5)){
  //   *status = 0;
  //   Serial.println(e);
```

```
  //   return;
  // }
}

void pidIndexMotor(int* setpoint, int *status){
// int myNum[3] = {10, 20, 30};
  // int motorChannel[2] = {2,3};
  int fingerAngle = encoderIndexCount;
  // Serial.println("pidFingerMotor running.. ");

    // int motorChannel[2] = {4,5};
    // Serial.println("Index finger moving");

  // int target_update = 0;
  // if (target_update == 0){
  //   // int target = 0;
  //   int target = fingerAngle + *setpoint;
  //   int e = target - fingerAngle;
  //   Serial.print("e = ");
  //   Serial.println(e);
  //   target_update ++;
  // }
  // int target = fingerAngle + *setpoint;

  int e = *setpoint-fingerAngle;
  // Serial.print("Error:");
  // Serial.println(e);
  // Serial.print("fingerAngle = ");
  // Serial.println(fingerAngle);


  // if (fingerAngle > 1800){
  //   *status = 0;
  //   return;
  // }

  // if ((e > 8) || (e < -8)){

  long currT = micros();

  float deltaT = ((float)(currT-prevT))/1.0e6;
  prevT = currT;

  // int e = *setpoint-fingerAngle;
  // Serial.print("setpoint = ");
  // Serial.println(*setpoint);
  // Serial.print("fingerAngle = ");
  // Serial.println(fingerAngle);

  float dedt = (e-eprev)/(deltaT);

  eintegral = eintegral + e*deltaT;

  // Control signal U

  float u = Kp_index*e + Kd_index*dedt + Ki_index*eintegral;
  Serial.println(u);

  // motor power
  float pwr = fabs(u);
  if (pwr>250){
    pwr=250;
  }

  int dir = 1;
  if (u<0){
    dir = -1;
  }


  int moto1 = motorChannel[4];
```

```
  int moto2 = motorChannel[5];
    // Serial.print("Moto1: ");Serial.println(moto1);
    // Serial.print("Moto2: ");Serial.println(moto2);
  setMotor(moto1, moto2, dir,pwr);


  eprev = e;

}

void pidThumbMotor(int* setpoint, int *status){
// int myNum[3] = {10, 20, 30};
  // int motorChannel[2] = {2,3};
  int fingerAngle = encoderThumbCount;
  // Serial.println("pidFingerMotor running.. ");

    // int target_update = 0;
  // if (target_update == 0){
  //   // int target = 0;
  //   int target = fingerAngle + *setpoint;
  //   int e = target - fingerAngle;
  //   Serial.print("e = ");
  //   Serial.println(e);
  //   target_update ++;
  // }
  // int target = fingerAngle + *setpoint;

  int e = *setpoint-fingerAngle;
  // Serial.print("Error:");
  // Serial.println(e);
  // Serial.print("fingerAngle = ");
  // Serial.println(fingerAngle);


  // if (fingerAngle > 1800){
  //   *status = 0;
  //   return;
  // }

  // if ((e > 8) || (e < -8)){

  long currT = micros();

  float deltaT = ((float)(currT-prevT))/1.0e6;
  prevT = currT;

  // int e = *setpoint-fingerAngle;
  // Serial.print("setpoint = ");
  // Serial.println(*setpoint);
  // Serial.print("fingerAngle = ");
  // Serial.println(fingerAngle);

  float dedt = (e-eprev)/(deltaT);

  eintegral = eintegral + e*deltaT;

  // Control signal U

  float u = Kp_thumb*e + Kd_thumb*dedt + Ki_thumb*eintegral;
  Serial.println(u);

  // motor power
  float pwr = fabs(u);
  if (pwr>255){
    pwr=255;
  }

  int dir = 1;
  if (u<0){
    dir = -1;
  }
```

```
  int moto1 = motorChannel[2];
  int moto2 = motorChannel[3];
    // Serial.print("pwr = ");
    // Serial.println(pwr);
    // Serial.print("Moto1: ");Serial.println(moto1);
    // Serial.print("Moto2: ");Serial.println(moto2);
  setMotor(moto1, moto2, dir,pwr);



  eprev = e;

}

  // if ((e < 5) && (e > -5)){
  //   *status = 0;
  //   Serial.println(e);
  //   return;
  // }
  // return;


void pidPinch(int* setpoint_index, int*setpoint_thumb, int *status){
  // int myNum[3] = {10, 20, 30};
  // int motorChannel[2] = {2,3};
  int ThumbAngle = encoderThumbCount;
  int IndexAngle = encoderIndexCount;
  // Serial.println("pidFingerMotor running.. ");

    // int target_update = 0;
  // if (target_update == 0){
  //   // int target = 0;
  //   int target = fingerAngle + *setpoint;
  //   int e = target - fingerAngle;
  //   Serial.print("e = ");
  //   Serial.println(e);
  //   target_update ++;
  // }
  // int target = fingerAngle + *setpoint;

  int e_thumb = *setpoint_thumb-ThumbAngle;
  int e_index = *setpoint_index-IndexAngle;
  // Serial.print("Error:");
  // Serial.println(e);
  // Serial.print("fingerAngle = ");
  // Serial.println(fingerAngle);


  // if (fingerAngle > 1800){
  //   *status = 0;
  //   return;
  // }

  // if ((e > 8) || (e < -8)){

  long currT = micros();

  float deltaT = ((float)(currT-prevT))/1.0e6;
  prevT = currT;

  // int e = *setpoint-fingerAngle;
  // Serial.print("setpoint = ");
  // Serial.println(*setpoint);
  // Serial.print("fingerAngle = ");
  // Serial.println(fingerAngle);

  float dedt_thumb = (e_thumb-eprev_thumb)/(deltaT);
  float dedt_index = (e_index-eprev_index)/(deltaT);

  eintegral = 0;//eintegral + e*deltaT;
```

```

  // Control signal U

  float u_thumb = Kp_thumb*e_thumb + Kd_thumb*dedt_thumb + Ki_thumb*eintegral;
  float u_index = Kp_index*e_index + Kd_index*dedt_index + Ki_index*eintegral;
  // Serial.println(u);

  // motor power
  float pwr_thumb = fabs(u_thumb);
  float pwr_index = fabs(u_index);

  if (pwr_thumb>255){
    pwr_thumb=255;
  }
  if (pwr_index>255){
    pwr_index=255;
  }

  int dir_thumb = 1;
  if (u_thumb<0){
    dir_thumb = -1;
  }
  int dir_index = 1;
  if (u_index<0){
    dir_index = -1;
  }

  int motoT1 = motorChannel[2];
  int motoT2 = motorChannel[3];
  int motoI1 = motorChannel[4];
  int motoI2 = motorChannel[5];
    // Serial.print("pwr = ");
    // Serial.println(pwr);
    // Serial.print("Moto1: ");Serial.println(moto1);
    // Serial.print("Moto2: ");Serial.println(moto2);
  setMotor(motoT1, motoT2, dir_thumb, pwr_thumb);
  setMotor(motoI1, motoI2, dir_index, pwr_index);



  eprev_thumb = e_thumb;
  eprev_index = e_index;

}

void pidHandAndFingers(int* setpoint_hand, int* setpoint_index, int*setpoint_thumb, int *
    status){
  // int myNum[3] = {10, 20, 30};
  // int motorChannel[2] = {2,3};
  checkQuadrant(readAngle());
  int HandAngle = totalAngle;
  int ThumbAngle = encoderThumbCount;
  int IndexAngle = encoderIndexCount;
  // Serial.println("pidFingerMotor running.. ");

    // int target_update = 0;
  // if (target_update == 0){
  //   // int target = 0;
  //   int target = fingerAngle + *setpoint;
  //   int e = target - fingerAngle;
  //   Serial.print("e = ");
  //   Serial.println(e);
  //   target_update ++;
  // }
  // int target = fingerAngle + *setpoint;
  int e_hand =  *setpoint_hand-HandAngle;
  int e_thumb = *setpoint_thumb-ThumbAngle;
  int e_index = *setpoint_index-IndexAngle;
  // Serial.print("Error:");
  // Serial.println(e);
  // Serial.print("fingerAngle = ");
```

```
// Serial.println(fingerAngle);


// if (fingerAngle > 1800){
//   *status = 0;
//   return;
// }

// if ((e > 8) || (e < -8)){

long currT = micros();

float deltaT = ((float)(currT-prevT))/1.0e6;
prevT = currT;

// int e = *setpoint-fingerAngle;
// Serial.print("setpoint = ");
// Serial.println(*setpoint);
// Serial.print("fingerAngle = ");
// Serial.println(fingerAngle);
float dedt_hand =  (e_hand - eprev_hand)/(deltaT);
float dedt_thumb = (e_thumb-eprev_thumb)/(deltaT);
float dedt_index = (e_index-eprev_index)/(deltaT);

eintegral = 0;//eintegral + e*deltaT;

// Control signal U
float u_hand  = Kp_hand *e_hand  + Kd_hand *dedt_hand  + Ki_hand *eintegral;
float u_thumb = Kp_thumb*e_thumb + Kd_thumb*dedt_thumb + Ki_thumb*eintegral;
float u_index = Kp_index*e_index + Kd_index*dedt_index + Ki_index*eintegral;
// Serial.println(u);

// motor power
float pwr_hand = fabs(u_hand);
float pwr_thumb = fabs(u_thumb);
float pwr_index = fabs(u_index);

if (pwr_hand>200){
  pwr_hand=200;
}
if (pwr_thumb>255){
  pwr_thumb=255;
}
if (pwr_index>255){
  pwr_index=255;
}

int dir_hand = 1;
if (u_hand <0){
  dir_hand = -1;
}
int dir_thumb = 1;
if (u_thumb<0){
  dir_thumb = -1;
}
int dir_index = 1;
if (u_index<0){
  dir_index = -1;
}


int motoH1 = motorChannel[0];
int motoH2 = motorChannel[1];
int motoT1 = motorChannel[2];
int motoT2 = motorChannel[3];
int motoI1 = motorChannel[4];
int motoI2 = motorChannel[5];
  // Serial.print("pwr = ");
  // Serial.println(pwr);
  // Serial.print("Moto1: ");Serial.println(moto1);
  // Serial.print("Moto2: ");Serial.println(moto2);
```

```
  setMotor(motoH1, motoH2, dir_hand, pwr_hand);
  setMotor(motoT1, motoT2, dir_thumb, pwr_thumb);
  setMotor(motoI1, motoI2, dir_index, pwr_index);


  eprev_hand  = e_hand;
  eprev_thumb = e_thumb;
  eprev_index = e_index;


  // if (counter == 80){
  Serial.print("E_hand: ");Serial.println(e_hand);//Serial.print("  E_index: ");Serial.print(e_index);Serial.print("  E_thumb: ");Serial.println(e_thumb);
  // Serial.print("Position: ");Serial.println(HandAngle);
    // counter = 0;
  // }
  // counter++;

}


void setMotor(int moto1, int moto2, int dir, int pwmVal){
  if (dir==1){
    ledcWrite(moto1, pwmVal);
    ledcWrite(moto2, 0);
  }
  else if (dir == -1){
    ledcWrite(moto1, 0);
    ledcWrite(moto2, pwmVal);
  }
  else{
    ledcWrite(moto1, 0);
    ledcWrite(moto2, 0);
  }


}
// void pidPosition(int* setpoint, float* error, int *status){

//       oldTime = millis();
//       checkQuadrant(readAngle());

//       if(totalAngle<-720){
//         *status = 0;
//         return;
//       }


//       *error = *setpoint - totalAngle;
//       while (*error > 10){
//         float proportional = PIDParams.Kp * (*error);

//         PIDParams.integrator = PIDParams.integrator + 0.5f * PIDParams.Ki * PIDParams.T * ((*error) + PIDParams.prevError);

//       /* Anti-wind-up via integrator clamping */
//         if (PIDParams.integrator > PIDParams.limMaxInt) {
//             PIDParams.integrator = PIDParams.limMaxInt;
//         } else if (PIDParams.integrator < PIDParams.limMinInt) {
//             PIDParams.integrator = PIDParams.limMinInt;
//         }
//       /*
//       * Derivative (band-limited differentiator)
//       */
//         // PIDParams.differentiator = -(2.0f * PIDParams.Kd * (totalAngle - PIDParams.prevMeasurement)       /* Note: derivative on measurement, therefore minus sign in front of equation! */
//         //                        + (2.0f * PIDParams.tau - PIDParams.T) * PIDParams.differentiator) / (2.0f * PIDParams.tau + PIDParams.T);
//       //Serial.println(PIDParams.integrator);
//       /*
```

```
//        * Compute output and apply limits
//        */
//          PIDParams.out = proportional + PIDParams.integrator;//int(proportional + PIDParams
    .integrator + PIDParams.differentiator);
//          if (PIDParams.out > PIDParams.limMax) PIDParams.out = int(PIDParams.limMax);
//          else if (PIDParams.out < PIDParams.limMin) PIDParams.out = int(PIDParams.limMin);

//          // Serial.print("PID Output: ");Serial.println(PIDParams.out);
//          // if(abs(PIDParams.out) < 10) *setpoint = totalAngle;
//        /* Store error and measurement for later use */
//          PIDParams.prevError = *error;
//          //Serial.print("Speed: "); Serial.println(speed);
//          //if(abs(speed) < 1) PIDParams.out = 0;
//          if(abs(PIDParams.out) < 10){
//            PIDParams.out = 0;
//            }
//          PIDParams.prevMeasurement = totalAngle;
//          if(PIDParams.out<0){
//            ledcWrite(cw, abs(PIDParams.out));
//            ledcWrite(ccw, 0);
//            }
//          else if (PIDParams.out>0){
//            ledcWrite(cw, 0);
//            ledcWrite(ccw, PIDParams.out);
//            }
//          else{
//            ledcWrite(cw, 0);
//            ledcWrite(ccw, 0);
//          }
//        // Serial.print("Output: "); Serial.println(PIDParams.out);
//        // Serial.print("Error: "); Serial.println(PIDParams.prevError);

//          // speed = totalAngle-PIDParams.prevMeasurement;
//          checkQuadrant(readAngle());
//          }
//          // Serial.print("Error: "); Serial.println(error);
//          // Serial.print("Position: "); Serial.println(totalAngle);


// }

void initialize(){

  pinMode(mainMotorPin1,  OUTPUT);
  pinMode(mainMotorPin2,  OUTPUT);
  pinMode(thumbMotorPin1, OUTPUT);
  pinMode(thumbMotorPin2, OUTPUT);
  pinMode(indexMotorPin1, OUTPUT);
  pinMode(indexMotorPin2, OUTPUT);

  pinMode(EncoderThumbA, INPUT);
  pinMode(EncoderThumbB, INPUT);
  pinMode(EncoderIndexA, INPUT);
  pinMode(EncoderIndexB, INPUT);


  ledcSetup(0, 20000, 8);         // Setup for all channels
  ledcSetup(1, 20000, 8);
  ledcSetup(2, 16000, 8);
  ledcSetup(3, 16000, 8);
  ledcSetup(4, 20000, 8);
  ledcSetup(5, 20000, 8);

  ledcAttachPin(mainMotorPin1,  0);
  ledcAttachPin(mainMotorPin2,  1);
  ledcAttachPin(thumbMotorPin1, 2);
  ledcAttachPin(thumbMotorPin2, 3);
  ledcAttachPin(indexMotorPin1, 4);
  ledcAttachPin(indexMotorPin2, 5);

  delay(100);
```

```
  Serial.println("Status:␣Motor␣pins␣assigned.");
  delay(10);

  int controlMode = 1;
  Serial.println("Status:␣Connecting␣encoder..");
  delay(10);
  I2Cone.begin(21,22);
  encoder.setI2CInstance(&I2Cone);
  uint8_t new_address=0x0A;
  encoder.set_address(new_address);
  encoder.default_cfg();
  bool encoder_connected = false;
  deleteTask = false;

  attachInterrupt(digitalPinToInterrupt(EncoderThumbA), updateEncoderThumb, CHANGE);
  // attachInterrupt(digitalPinToInterrupt(EncoderThumbB), updateEncoderThumb, CHANGE);
  attachInterrupt(digitalPinToInterrupt(EncoderIndexA), updateEncoderIndex, CHANGE);
  // attachInterrupt(digitalPinToInterrupt(EncoderIndexB), updateEncoderIndex, CHANGE);
  last_state_IA = digitalRead(EncoderIndexA);
  last_state_TA = digitalRead(EncoderThumbA);

  while(!encoder_connected){
    if (encoder.check_communication()==C3DHALL11_OK){
      Serial.println("Status:␣Encoder␣connected");
      delay(100);
      encoder_connected = true;
      //sensor[i].offsetCalibration();
    } else{
      Serial.println("Status:␣Encoder␣error");
      delay(1000);
    }
  }

 xTaskCreate(
                    readPositionTask,   /* Function to implement the task */
                    "ReadPosition", /* Name of the task */
                    10000,      /* Stack size in words */
                    (void *)&deleteTask,       /* Task input parameter */
                    20,          /* Priority of the task */
                    &xHandle       /* Task handle. */
                    );  /* Core where the task should run */

  initPosition(&deleteTask);

  // if(controlMode == 0){
  //   connect_Myo();
  //   xTaskCreatePinnedToCore(
  //                   MyoDisconnectTask,   /* Function to implement the task */
  //                   "MyoDisconnectTask", /* Name of the task */
  //                   10000,      /* Stack size in words */
  //                   NULL,       /* Task input parameter */
  //                   0,          /* Priority of the task */
  //                   NULL,       /* Task handle. */
  //                   0);  /* Core where the task should run */

  // xTaskCreatePinnedToCore(
  //                     processEMGData,   /* Function to implement the task */
  //                     "processEMGData", /* Name of the task */
  //                     10000,      /* Stack size in words */
  //                     NULL,       /* Task input parameter */
  //                     1,          /* Priority of the task */
  //                     NULL,       /* Task handle. */
  //                     0);  /* Core where the task should run */

  // xTaskCreatePinnedToCore(
  //                     MotorControlTask,   /* Function to implement the task */
  //                     "MotorControlTask", /* Name of the task */
  //                     10000,      /* Stack size in words */
  //                     NULL,       /* Task input parameter */
  //                     2,          /* Priority of the task */
  //                     NULL,       /* Task handle. */
```

```
  //                      0);  /* Core where the task should run */
  // }
  initialized = true;
}

void setup()
{
  Serial.begin(230400);
  delay(10);
  xTaskCreatePinnedToCore(
                processSerialData,   /* Function to implement the task */
                "SerialData", /* Name of the task */
                10000,      /* Stack size in words */
                NULL,       /* Task input parameter */
                10,          /* Priority of the task */
                NULL,       /* Task handle. */
                0);  /* Core where the task should run */

}


void loop()
{

}
```

B Appendices

Table B.1: Specifications of the Pololu miniature brushed DC metal gearmotor with a gearbox cross-section of 10×12 mm and a 9 mm long, 3 mm diameter D-shaped gearbox output shaft.

Motor specifications	
Size	10 x 12 x 25 mm
Gear ratio	75.81:1
No-load speed @ 6V	410 rpm
Stall current @ 6V	1.6 A
Stall torque @ 6V	.13 Nm
Max output power @ 6V	1.4 W
Max efficiency @ 6V	40 %
Speed at max efficiency	340 rpm
Torque at max efficiency	0.023 Nm
Current at max efficiency	0.34 A
Output power at max efficiency	0.80 W

Table B.2: Specifications of the Maxon DCX19s motor with a 62:1 GPX19 gearbox

Motor specifications	
Size	19 x 68.7 mm
Gear ratio	62:1
No-load speed @ 12V	12700 rpm
Stall current @ 12V	8.26 A
Stall torque @ 12V	.735 Nm
Max output power @ 12V	16.1 W
Max efficiency @ 12V	82.2 %
Speed at max efficiency	1080 rpm
Torque at max efficiency	0.114 Nm
Current at max efficiency	1.35 A
Output power at max efficiency	16.1 W