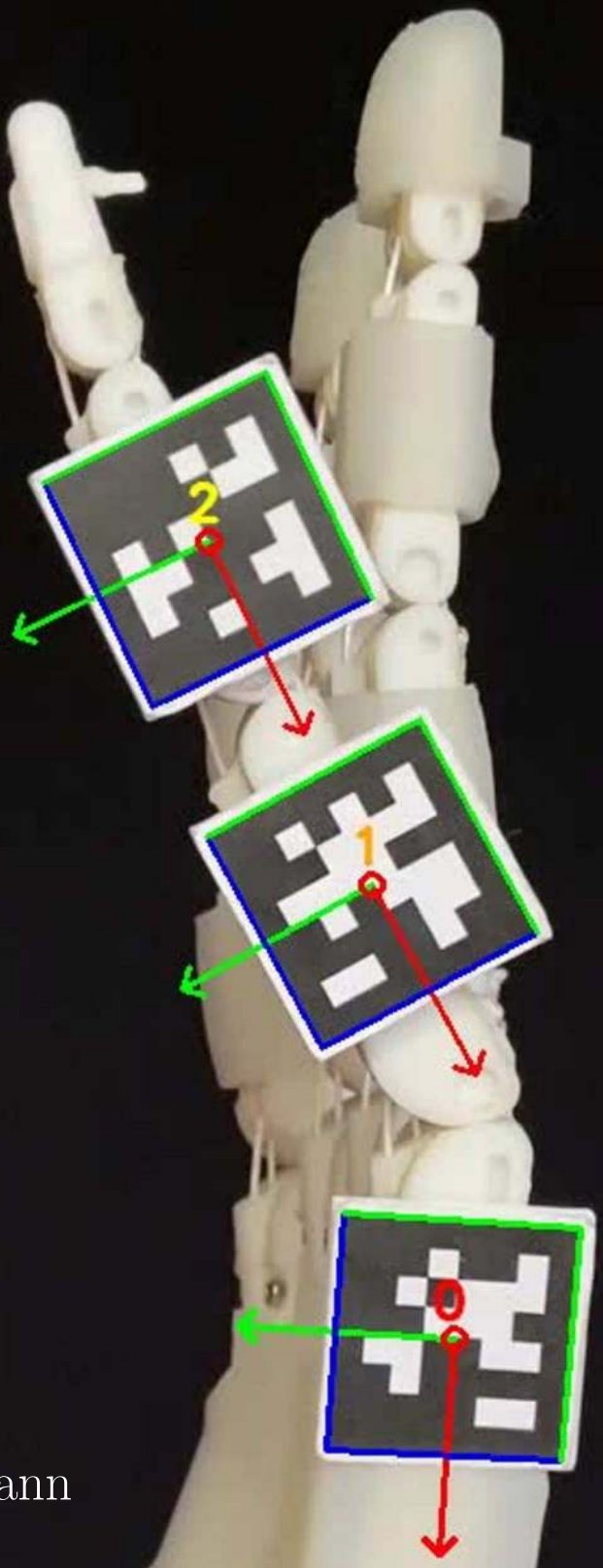
ETH zürich

Soft Robotics Laboratory

Bachelor's Thesis

Dynamic Dexterous Manipulation



Maximilian Eberlein

August 2023

Supervision

Prof. Dr. Robert Katzschmann
Benedek Forrai

Abstract

Humans possess exceptional capabilities in manipulating objects with both dexterity and speed, enabling them to execute actions precisely and swiftly. However, the realm of *high-speed*, skillful object manipulation performed by robots remains largely unexplored within the research community. This thesis takes the initial step in addressing this gap by teaching an anthropomorphic robotic hand the rapid, dexterous skill of dynamically throwing and catching a ball. This task diverges significantly from the conventional methods of slower in-palm object reorientation that have received attention in recent works. Given the need for increased precision and predictability in fast manipulation tasks, the thesis first focuses on a comprehensive system identification of the robot, with a specific emphasis on benchmarking its latency and proprioception accuracy. Subsequently, the research explores the process of shaping rewards to encourage fast behavior in simulation-based training. Furthermore, it addresses the main challenges arising during the transition from simulation to the real-world setup due to the high-speed actions involved. The thesis concludes by presenting the main findings and challenges encountered in the realm of fast, dexterous manipulation.

Contents

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Thesis Objectives . . . 1
1.3 Related Work . . . 2

2 Preparation **7**
2.1 Preparatory Work . . . 7
2.2 Outline . . . 9

3 Benchmarking the Faive Hand **9**
3.1 Initial Setup . . . 9
3.2 Computer Vision for Groundtruth Data . . . 10
3.3 Tracking Markers . . . 11
3.4 Tracking Apriltags . . . 16
3.5 Benchmarking Latency . . . 21
3.6 Benchmarking Proprioception Accuracy . . . 26

4 Training in Simulation **30**
4.1 IsaacGym Acquaintance & Reward Shaping . . . 30
4.2 Latency Curriculum Implementation & Open-Loop vs. Closed-Loop . . . 32

5 Initial Stages of Sim-To-Real **37**
5.1 Demonstrating Physical Feasibility . . . 37
5.2 Object Tracking for Closed-Loop Control . . . 39
5.3 Tuning of Tendon Dynamics in Simulation . . . 45

6 Conclusion **49**

7 Acknowledgements **50**

A Appendices **52**
A.1 Angle Measurement using Markers and iPhone . . . 52
A.2 Oak -D S2: Camera module specifications . . . 53
A.3 Intrinsic Matrix of iPhone 13 Pro . . . 53
A.4 Intrinsic Matrix of Oak -D S2 MonoCamera . . . 53
A.5 Oak -D S2 120 FPS Videofeed Publisher Script . . . 54

1 Introduction

1.1 Motivation

The manipulation of objects with dexterity and speed is a fundamental skill possessed by humans, enabling us to perform a wide range of tasks fast and efficiently. As efforts are made to elevate the sophistication of robots and their ability to solve complex tasks, a growing interest in closing the disparity between robots' object manipulation skills and those of humans has become evident. In particular, great focus in the research community has been on the reproduction of the intricate capabilities of the human hand, which have empowered humans for centuries to manipulate objects in diverse ways. While significant efforts have been made in the development of modern robotic hands to achieve advanced dexterity, their speed in executing complex manipulations has remained an area of limited exploration to date. However, to truly replicate human object manipulation skills, addressing speed is also crucial.

The largely unexplored potential of leveraging robot speed for dexterous manipulation tasks is attributed to the inherent complexities of such manipulations. A striking example of this is the in-hand reorientation of objects, a task seemingly effortless for humans yet posing intricate challenges for robots due to the complexities of high-dimensional observation and action spaces, as well as discontinuous contact patterns. As a result, achieving maximal robustness necessitates that current state-of-the-art object reorientation is performed noticeably more slowly by robots compared to humans, exemplified by Handa et al. (2022) who achieved successful object reorientation in 6-7 seconds, in contrast to the 1-2 seconds typically taken by a human.[1]

1.2 Thesis Objectives

This thesis aims to build on the capabilities of the anthropomorphic Faive hand[2] developed at the Soft Robotics Lab of ETH Zurich (Fig. 1) to fit the current state-of-the-art approaches for dexterous manipulation to the uncharted territory of fast, dynamic dexterous manipulation.

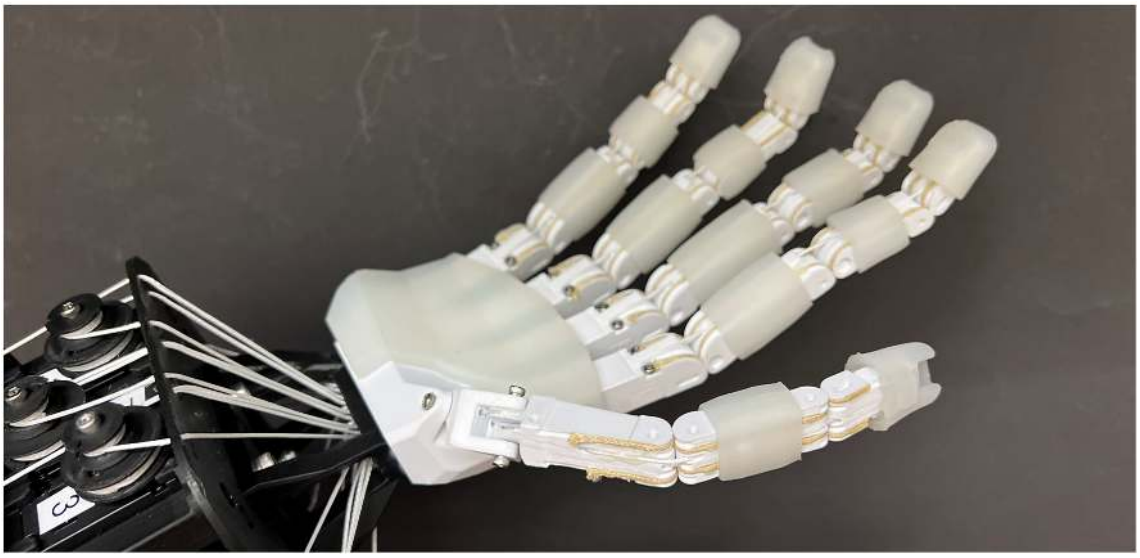


Figure 1: The Faive Hand, developed at the Soft Robotics Lab, is a 3D-printed, low-cost biomimetic tendon-driven hand featuring rolling contact joints.

The Faive hand possesses a distinct advantage over humans in being able to fully open and close at a frequency of 2.5 Hz. This capability is leveraged by employing Reinforcement Learning (RL) techniques to *dynamically* manipulate a ball through throwing and catching, which contrasts with

[1] Value obtained from my own hands-on experiments.
[2] https://www.faive-robotics.com/, 2023.

the *slower* palm-based reorientation manipulation demonstrated in recent studies. The objective is to showcase that learning dynamic dexterous behaviors can lead to efficient task-solving mechanisms that leverage the robot's speed while maintaining robustness.

To achieve this goal, several prerequisites for learning fast behaviors must be taken into account. First, an evaluation of the physical hand's latency and proprioception accuracy is essential. Excessive latency between action commands and their execution can severely hinder the implementation of closed-loop high-speed behaviors. Furthermore, incorporating latency into the hand's simulation model for training can be crucial to bridge the gap between simulation and reality.

Additionally, the significance of proprioception accuracy cannot be underestimated. For successful object throwing and catching without the capability to reposition the hand, a highly precise and robust throwing mechanism is essential to ensure that the ball consistently lands within the hand's palm. This necessitates accurate proprioception of the state of the fingers.

After the system identification process, the hand undergoes simulation-based training. The goal is to identify an optimal policy that reliably performs the ball throw-and-catch task through multiple iterations of intelligent reward shaping. Subsequently, the focus shifts to addressing the challenging sim-to-real gap for fast manipulation tasks and preparing the transfer of the learned behavior from simulation to the real hand.

Ultimately, an assessment of the feasibility and challenges associated with teaching a multi-fingered robot hand the skills for a fast dexterous manipulation task is provided, along with the presentation of the final results.

1.3 Related Work

This thesis heavily draws upon the valuable insights into the training and performance of robot agents in the context of complex dexterous tasks. Previous research has explored various directions, encompassing different robot hardware, reinforcement and imitation learning methods, manipulation tasks, and state space observation techniques. Initially, attention is directed towards studies that have examined dexterous manipulation capabilities in both simulated environments and real-world robotic systems: Handa et al. (2022), for instance, used a RL-based approach to reorientate a cube with a 4-fingered wrist-locked Allegro hand (Fig. 2).



Figure 2: DeXtreme: Transfer of Agile In-hand Manipulation from Simulation to Reality

While their results were very robust, they penalized movements that were too quick and the duration taken by the robot to reorient the cube was not a primary concern as long as the objective was successfully accomplished. In contrast, the goal in this thesis is to encourage fast behavior. Another relevant study by Qin et al. (2023) proposed a sim-to-real framework for dexterous manipulation. Their approach demonstrated the ability to generalize to new objects of the same category in the real world. Using an XArm6 and Allegro hand, they successfully grasped objects and opened doors by leveraging imagined hand point clouds (Fig. 3).

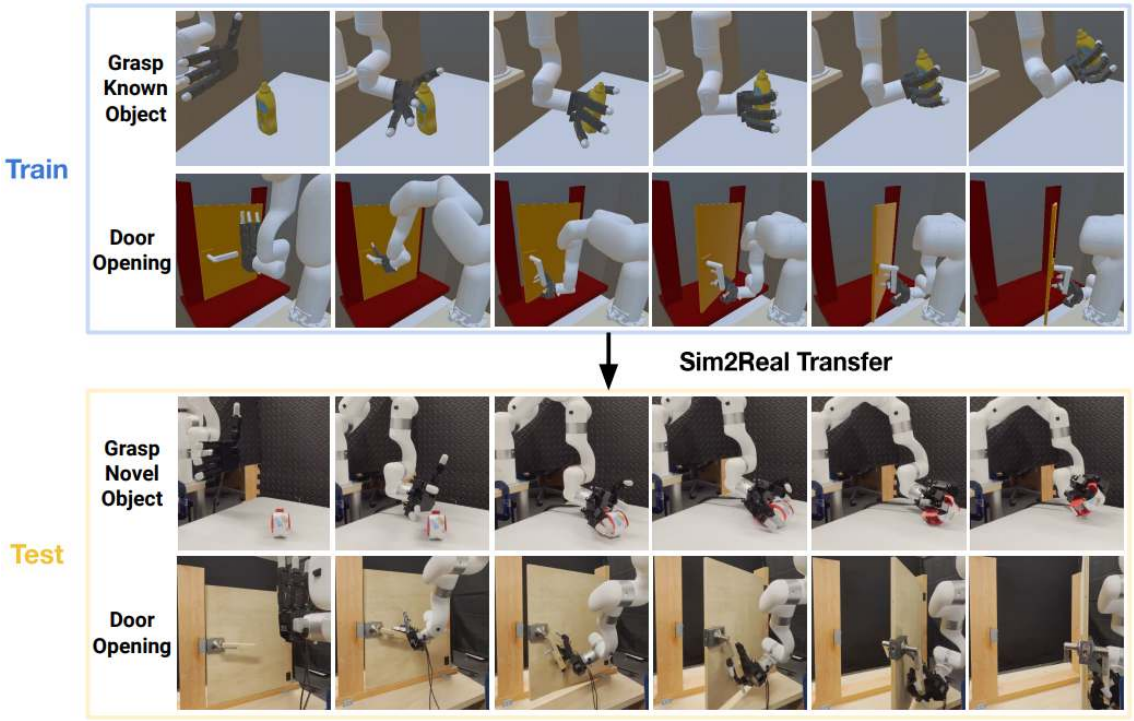
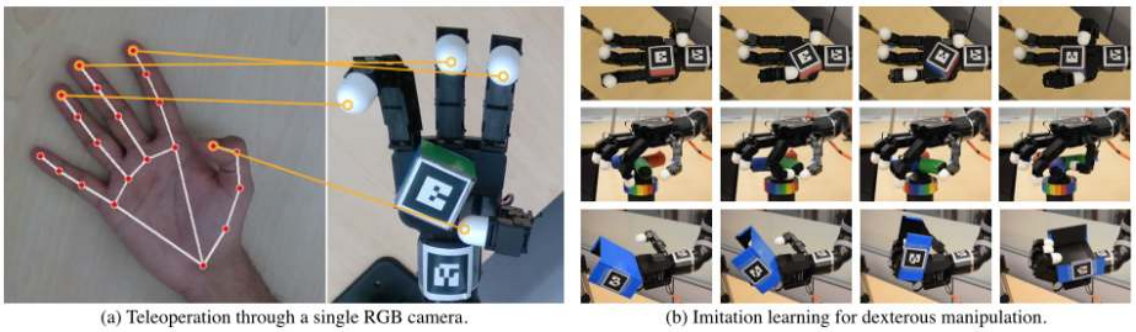


Figure 3: DexPoint: Generalizable Point Cloud Reinforcement Learning for Sim-to-Real Dexterous Manipulation

Similarly to the previous case, the emphasis in this work was on robustly solving dexterous tasks without considering speed as a primary factor. It is worth noting that both of these works utilized pure RL with Proximal Policy Optimization (PPO) as their learning approach. However, it is important to highlight that other research efforts have explored the advantages of Imitation Learning, which offers improved sample efficiency and enables the learning of more natural behavior. For example, Arunachalam et al. (2022) proposed a novel imitation learning framework called DIME. By leveraging demonstrations collected of a human operator using a simple hand tracker, they were able to successfully solve complex manipulation tasks, such as flipping, spinning, and rotating specific objects, using an Allegro hand (Fig. 4).



(a) Teleoperation through a single RGB camera.

(b) Imitation learning for dexterous manipulation.

Figure 4: Dexterous Imitation Made Easy: A Learning-Based Framework for Efficient Dexterous Manipulation

Additionally, Shaw et al. (2023) developed a learning algorithm that leverages visual, action, and physical priors from in-domain demonstrations similar to those used in the DIME framework. Moreover, they introduced a novel training data input component by incorporating human videos into their learning pipeline. This algorithm enabled an xArm6 and LEAP hand to learn various manipulation tasks, such as rotating, covering, uncovering, opening, picking, and placing diverse objects (Fig. 5).

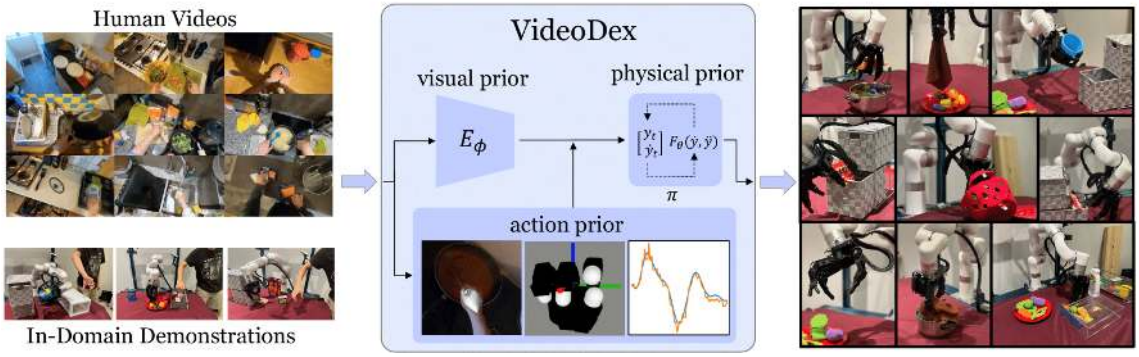


Figure 5: VideoDex: Learning Dexterity from Internet Videos

Zhao et al. (2023) conducted research specifically on fine manipulation tasks, such as threading cable ties or slotting a battery. To collect demonstrations for these tasks, they utilized a custom teleoperation interface. With the help of this interface, they successfully performed fine-grained manipulations using a bimanual system comprising two ViperX arms equipped with custom grippers (Fig. 6).

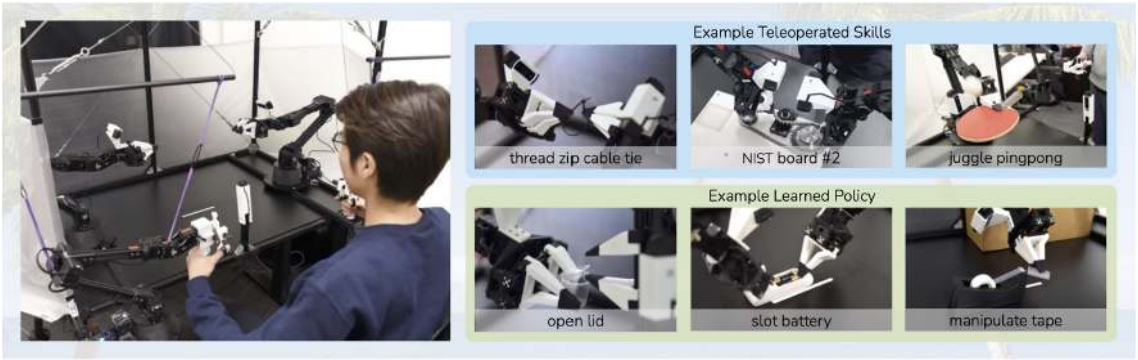


Figure 6: Learning Fine-Grained Bimanual Manipulation with Low-Cost Hardware

A common characteristic among all the aforementioned imitation learning approaches is their focus on achieving natural and robust behavior without emphasizing speed. Their primary goal is to solve tasks successfully while achieving higher sample efficiency through imitation learning. However, this thesis places its emphasis on the exclusive utilization of pure RL approaches. The objective is not to mimic human behavior as naturally as possible but to discover an innovative and dynamic solution for a dexterous manipulation task. Due to parallel GPU-accelerated calculations, low sample efficiency can be disregarded, rendering the use of imitation learning unnecessary for this research. As a result, the manual shaping of rewards emerges as a key factor in learning successful policies.

Alongside these studies, another intriguing proposal was introduced by Yin et al. (2023), which employed a pure RL approach utilizing an Allegro hand without vision. They successfully rotated arbitrary objects around a single axis using only 16 contact sensors (Fig. 7).

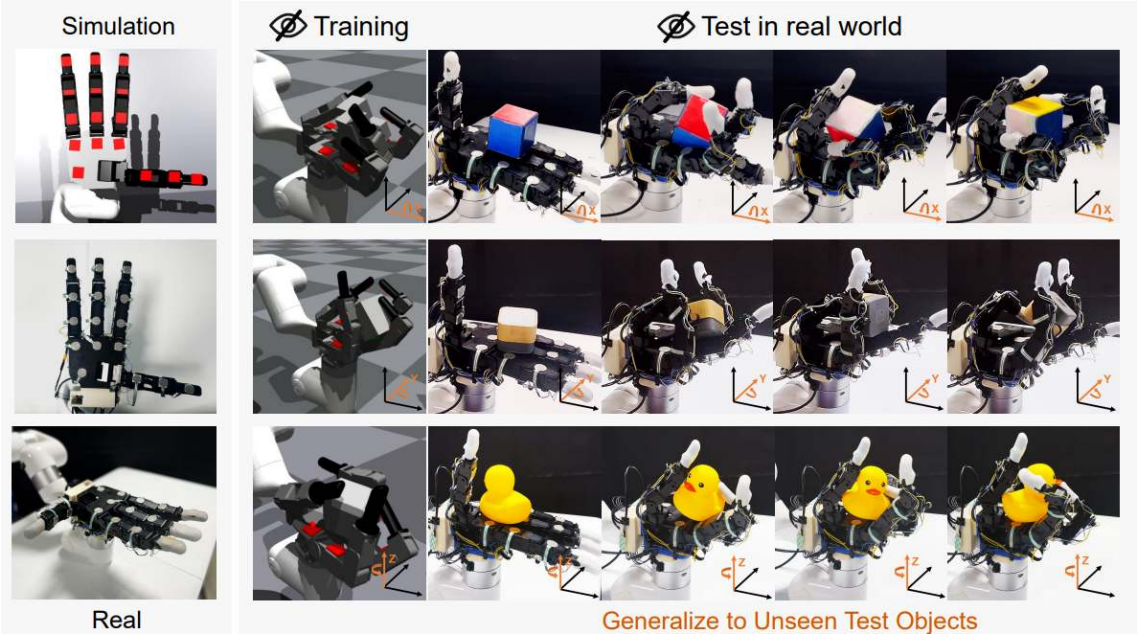


Figure 7: Rotating without Seeing: Towards In-hand Dexterity through Touch

As the ball leaves the hand during throwing, maintaining continuous contact becomes impossible. Consequently, this thesis prioritizes vision-based techniques to observe the ball's state, without pursuing alternative approaches such as relying solely on contact sensors.

While the primary objective of this thesis is to enable the deployment of fast, dexterous policies on a real-world robotic framework, it is worth noting that some research works have presented results solely in simulation without addressing the challenges of sim-to-real transfer. However, these works still provide valuable insights into training in simulation and reward shaping, which hold significance for this thesis. In their pioneering work, Rajeswaran et al. (2018) demonstrated the capability of model-free RL approaches to successfully solve complex manipulation tasks, including object relocation, in-hand manipulation, and tool use with a simulated 5-fingered Adroit hand (Fig. 8).



Figure 8: Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations

Moreover, Qin et al. (2021) demonstrated in their work the training of an Adroit hand using

internet videos for imitation learning, enabling the hand to acquire dexterous manipulation skills such as object relocation and pouring fluids into a cup (Fig. 9).

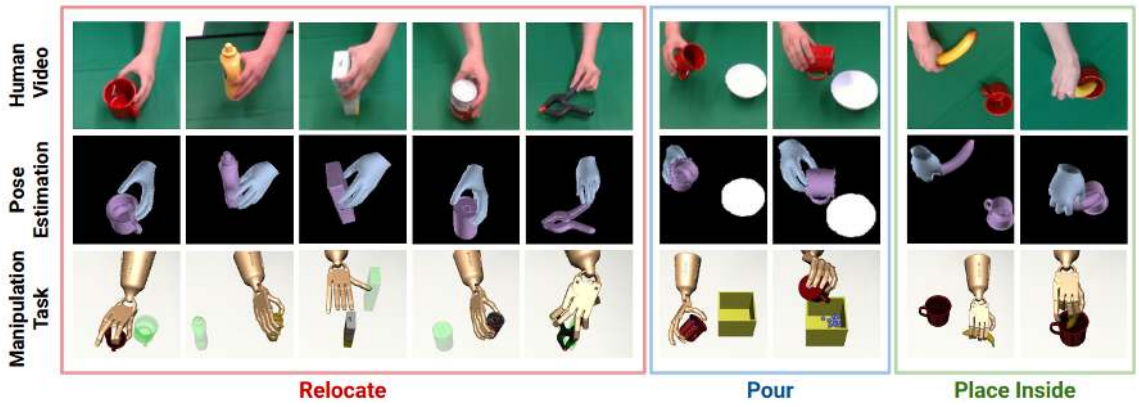


Figure 9: DexMV: Imitation Learning for Dexterous Manipulation from Human Videos

Both Chen et al. (2021) and Huang et al. (2021) employed a pure RL approach to manipulate objects using a simulated 5-fingered Shadow hand. The former utilized PPO to teach the hand to reorient arbitrary objects, even when the hand was facing downwards (Fig. 10). On the other hand, the latter utilized the Deep Deterministic Policy Gradient (DDPG) algorithm combined with Hindsight Experience Replay (HER) to rotate objects around one axis, with a strong emphasis on achieving high generalization to new objects with unseen shapes or sizes (Fig. 11).

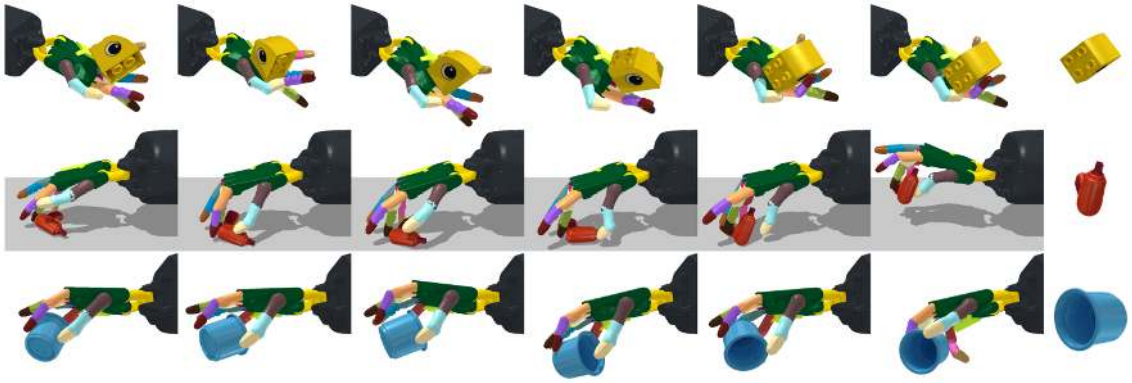


Figure 10: A System for General In-Hand Object Re-Orientation



Figure 11: Generalization in Dexterous Manipulation via Geometry-Aware Multi-Task Learning

In this thesis, the main goal is to catch and throw a tennis-sized ball, which means that achieving high generalization capability to new objects is not a primary objective. Nonetheless, despite

addressing diverse aspects and methods, all the cited works contribute to a fundamental understanding of the core challenges in dexterous manipulation and provide valuable insights into the learning process, especially in terms of structuring training, observing states, and shaping rewards for robot hands with complex, high-dimensional action spaces.

2 Preparation

2.1 Preparatory Work

In the initial phase of my thesis, I undertook a thorough literature review (see Section 1.3) to gain a more profound understanding of the current state of dexterous object manipulation achieved by robotic hands, along with the methodologies employed for these purposes. During my research, I additionally discovered past works, such as those from Senoo et al. (2009), which already more than 20 years ago demonstrated the capabilities of robotic hands to perform highly dynamic and complex tasks, such as swiftly dribbling a ball or skillfully throwing and catching a cell phone. Despite the impressive nature of these tasks, the behavior of their robots was hard-coded, lacking adaptability to new objects, varied environmental settings, or unforeseen disruptions. Consequently, their robot hands exhibited limited generalization capability and robustness due to their static programming. In modern robotics, our emphasis is on achieving generalized behavior. Our robots must possess the agility to adjust to diverse object geometries, weights, and surface textures, adapting their actions based on changing object properties and environmental conditions. While it might be feasible to hard-code a robotic hand to rotate a specific object, such as a sphere, by a certain degree, the real challenge lies in handling novel unseen objects like a cube just as well. In such cases, a statically programmed hand would likely falter in effectively reorienting the new geometry. Addressing this issue of generalization capability, recent studies have predominantly adopted RL or IL algorithms to train robotic hands in various manipulation tasks, even with unfamiliar or diverse objects.

Reinforcement Learning (RL) and Imitation Learning (IL) are two fundamental methods through which a robot "learns" its behavior. In RL, a robot learns by interacting with its environment and adapting its actions through trial and error based on the outcomes it experiences. On the other hand, IL involves the robot mimicking the behavior demonstrated by a skilled human, often through techniques like teleoperation or video learning. Both methods have distinct advantages and can also be combined. I decided to go with a pure RL approach because of two reasons.

First, since my goal was to train the hand for rapid behavior, the human demonstrations required for IL would have needed to be fast as well. Gathering a sufficient number of consistent demonstrations[3], then retargeting the human hand motions to the robot, and addressing the complexities of high-frequency human hand pose detection, would have significantly consumed time within my thesis.

The second reason why I did not use an IL approach was that my goal was not necessarily to imitate human behavior but rather to find a fast and dexterous behavior that could solve a specific task efficiently. I did not want to restrict the robot to behavior that feels natural to humans, but rather support creative and unique solutions.

Therefore, I opted for a pure RL approach, with the flexibility of potentially resorting to IL if standalone RL proved insufficient for successful learning. There are plenty of RL algorithms to select from, and the choice depends on the specific task, with certain options being more suitable than others. In Fig. 12[4], a tree structure illustrating the taxonomy of frequently employed RL algorithms is provided. As depicted in the figure, the subsequent decision following the choice of RL was whether to pursue a Model-Free or Model-Based RL approach.

Model-Free and Model-Based RL approaches differ in how they address the challenge of learning optimal behavior in a dynamic environment. In Model-Free RL, the focus is on learning optimal ac-

[3] Performing a task repeatedly at high speed while maintaining similarity is challenging. For instance, try tossing a tennis ball into the air and catching it without using your wrist—you'll notice variations in your movements throughout.

[4] https://spinningup.openai.com/, 2023.

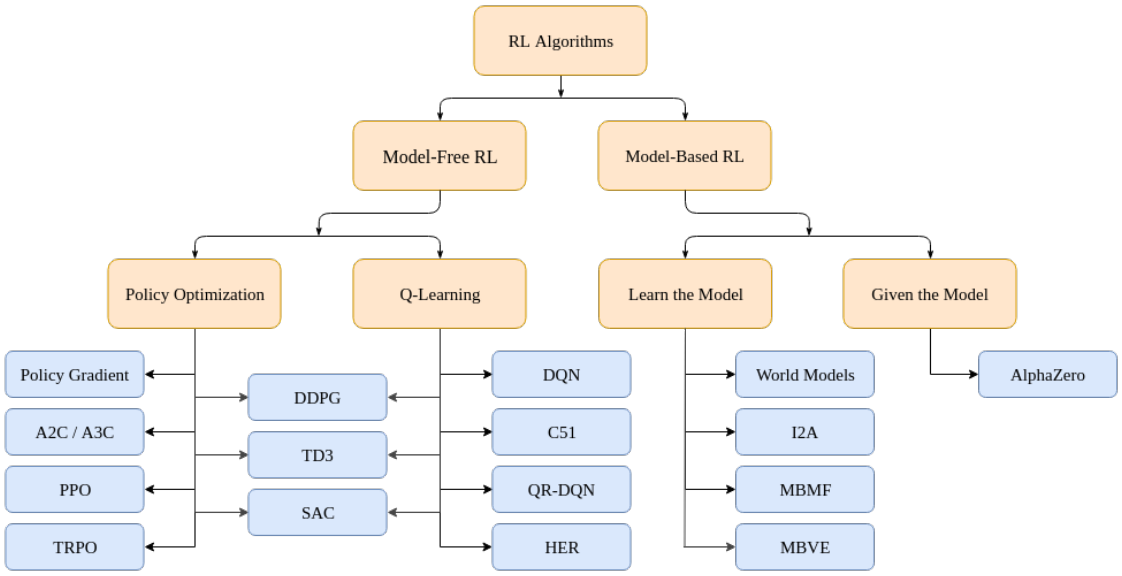


Figure 12: Overview of common Reinforcement Learning Algorithms

tions through direct experience, updating policies or values without explicit environment modeling. Conversely, Model-Based RL involves constructing an environment model to simulate trajectories and plan actions by learning transition dynamics for optimization. This approach proves useful when real-world interactions are costly or risky.

However, despite the availability of both approaches, the papers featured in Section 1.3 predominantly adopted a Model-Free RL approach. This preference for Model-Free RL can be attributed to the intricate nature of robotic hands, which possess numerous degrees of freedom (dof) due to their multiple independently actuated joints. In a model-based approach, the computation of action space to achieve specific behavior would rapidly escalate to infinity. Moreover, particularly in real-world scenarios, obtaining accurate dynamics models and state estimates for contact-rich, high-dimensional manipulation tasks is extremely challenging. Given these complexities, utilizing a model-free approach in high dof systems proves more practical and efficient, allowing the agent to learn optimal behavior within simulations without relying on a comprehensive model.

Since all the reviewed works employed Policy Optimization methods instead of Q-Learning for Model-Free RL, given their advantages in handling complex and high-dimensional action spaces, I chose to follow suit. As a result, my next step involved familiarizing myself with the most prevalent RL Policy Optimization techniques. To achieve this, I turned to the introductory Spinning Up RL Tutorial by OpenAI[4], enhancing my understanding of core RL concepts and delving into advanced terminology and Policy Optimization principles. I also explored the TRPO algorithm proposed by Schulman, Levine, et al. (2017), as well as the PPO algorithm proposed by Schulman, Wolski, et al. (2017). Both of these methods are sophisticated state-of-the-art techniques for policy optimization, well-suited for robotic applications. Given that PPO presents a more approachable alternative to TRPO with its clipping mechanism and notably widespread use among the dexterous manipulation literature I reviewed, I determined PPO to be the optimal choice for my RL approach. Furthermore, the Soft Robotics Lab had already developed a complete PPO implementation for the Faive hand, simplifying the usage of this methodology.

Having completed this preparatory research, the optimal approach for teaching dynamic dexterous manipulation to the Faive hand became evident. Subsequently, I needed to familiarize myself with PyTorch, a Python library for building and training neural networks. This was crucial for comprehending and building upon the existing PPO framework. To achieve this, I engaged in

PyTorch courses[5,6] and delved into the details of parallel training in simulation environments like IsaacGym or MuJoCo.

Considering that the Soft Robotics Lab was already equipped with a functional IsaacGym framework[7], my decision to work with it was straightforward. However, I also acknowledged the possibility of encountering challenges during the sim-to-real process, which could lead to exploring MuJoCo as a viable alternative if necessary.

Working with robotic systems, simulation environments, and specialized packages demands a high level of computer proficiency. Linux-based operating systems like Ubuntu are often preferred for such tasks due to their inherent adaptability, extensive customization options, and strong compatibility with a diverse range of tools commonly used in these domains. During my final preparatory phase, I learned to navigate and use Ubuntu, an operating system that was entirely new to me. I set up a dual-boot configuration with Ubuntu 20.04 on my laptop and followed an introductory Linux course[8], that provided me with essential insights into commands, practices, and fundamental Linux concepts that were crucial for my thesis.

2.2 Outline

After familiarizing myself with the relevant literature, RL methods, PyTorch, and Ubuntu, I embarked on the final step of the preparatory stage: outlining work packages and milestones for my thesis. The culmination of this effort resulted in the following timetable (Fig. 13):

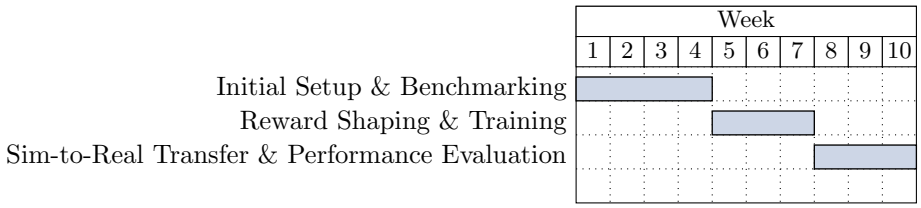


Figure 13: Approximate Timeline of the 10 weeks full-time thesis.

With these preparations in place, I was fully prepared to dedicate my time to achieving my main objective: enabling the Faive hand to perform the fast dexterous skill of throwing and catching a ball through system identification, RL learning, and sim-to-real gap adressing.

3 Benchmarking the Faive Hand

3.1 Initial Setup

As outlined in Section 1.1, the initial requirement for enabling fast behavior in the Faive hand in general was to benchmark its latency and proprioception accuracy. To benchmark these attributes, I needed to first familiarize myself with the physical mechanisms and properties of the hand. The Faive hand is a five-fingered tendon-driven anthropomorphic robotic hand, featuring 11 distinctively actuated rolling contact joints capable of flexion and extension. With its limitation in lateral finger movement, the hand possesses precisely 11 degrees of freedom (dof). The thumb of the Faive hand features 3 dof, while each of the other fingers has 2. In comparison, the 5-fingered Shadow Hand (20 dof) and the 4-fingered Allegro Hand (16 dof) possess more degrees of freedom due to their ability to perform both sagittal and lateral finger flexion. It is also important to highlight that the wrist of the Faive hand cannot be actuated; only its fingers are controllable. While humans commonly flex their wrist to throw up a ball and catch it, the Faive hand lacks this capability.[9] As I became acquainted with the hand's hardware, I also gained insights into the

[5] https://www.learnpytorch.io/, 2023.
[6] https://www.youtube.com/watch?v=c36lUUr864M, 2024.
[7] https://github.com/srl-ethz/faive_gym_oss, 2023.
[8] https://linuxjourney.com/, 2023.
[9] I considered attaching the hand to a robot arm to enable hand repositioning and reorientation, but this would have deviated from the main objectives of this thesis

movement of the fingers through antagonistic and agonistic tendons, along with learning how to properly retighten the tendons to prevent slack-related issues.

In addition to comprehending the hardware, I also needed to familiarize myself with the software and software-hardware interface within the Faive framework. To verify the functionality of the interface between my laptop and the Faive hand, I needed to delve into the source code managing the communication and control of the hand. After installing the required libraries and successfully resolving version incompatibilities, I read through the codebase to comprehensively analyze the communication protocols, hand calibration, control logic, and proprioception implementation employed within the Faive framework. During this phase, I encountered another essential toolset that I needed to learn for my work: the Robot Operating System (ROS). ROS is an open-source tool that enhances and simplifies interactions with robotic interfaces, and it is also a primary component of the Faive framework. Utilizing resources such as the comprehensive Beginner Level Core ROS Tutorial[10] and the ROS course offered by the ETH Robotics Systems Lab[11],"I acquired a solid understanding of working with ROS, establishing the required foundation for integrating ROS code into my application.

Armed with this foundational knowledge, I was well-prepared to embark on the initial main objective of my thesis: creating a user-friendly, compact, and independent framework designed for latency and proprioception accuracy benchmarking of the Faive hand.

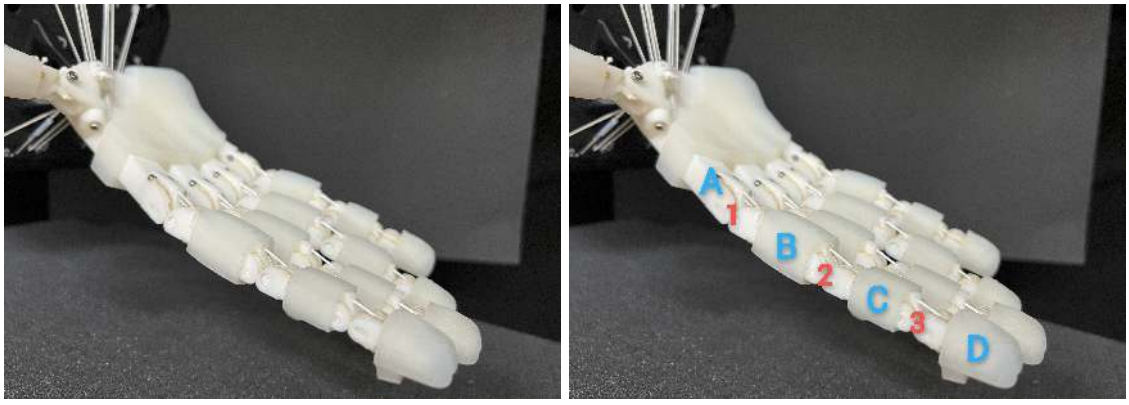
3.2 Computer Vision for Groundtruth Data

In order to benchmark the latency or proprioception accuracy of the Faive hand, I needed ground truth data about the hand's position at specific times. Since each finger is actuated using similar code logic, obtaining ground truth data for just one finger sufficed. Importantly, given my application's requirement for high-speed movements, capturing ground truth data at a high frame rate was essential. A low frame rate could result in insufficient positional data or imprecise measurements due to motion blur.

Fortunately, evaluating the ground truth data did not need to occur in real-time; it could be done after collecting all the measurements. This meant that I didn't need to stream the positional data of the Faive hand directly into sophisticated algorithms. Instead, I could log the data into a file for later evaluation. The main challenge was recording the actual position and velocity of the Faive hand's fingers at a high frequency.

My initial idea was to use a conventional Motion Capture (MoCap) System, which employs reflective spheres to track 3D objects and operates at 100 frames per second (FPS). However, there

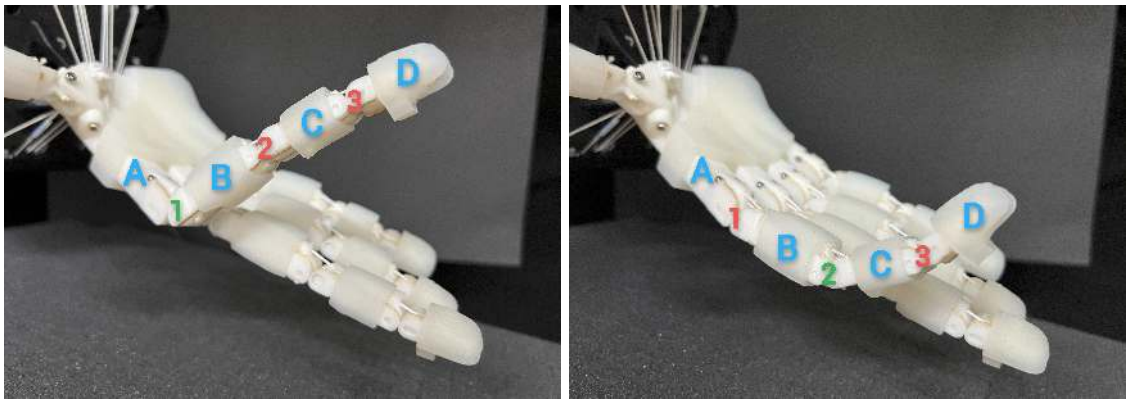
[10] http://wiki.ros.org/ROS/Tutorials, 2023.
[11] https://rsl.ethz.ch/education-students/lectures/ros.html, 2023.



(a) Side view of the Faive hand.

(b) All joints are relaxed.

Figure 14: The Faive index finger consists of four separate rigid bodies, denoted as A, B, C, and D, interconnected by rolling joints 1, 2, and 3.



(a) Joint 1 (proximal joint) is actuated.

(b) Joint 2 (distal joint) is actuated.

Figure 15: Only joints 1 and 2 can be controlled independently, while joint 3 is directly linked to the states of joints 1 and 2, making it dynamically dependent on them.

were two significant issues with this approach. The index finger of the Faive hand consists of four rigid segments (Fig. 14). While joint 3 between segments C and D cannot be controlled, the other joints 1 and 2 can be actuated independently (Fig. 15). As I was interested in the angle and angle change of the two independent joints 1 and 2, I needed to find a way to determine the orientation of segments A, B, and C. This was essential to accurately calculate both angles and their changes.

Conventional MoCap systems require at least three markers, each spaced at least 1 cm apart, per independent rigid body. Consequently, I would have had to attach a minimum of three markers, each with a diameter of at least 6.5 mm, to every segment without obstructing the cameras' view. This task alone would have presented a significant challenge. Moreover, attaching or screwing these spheres onto the hand in a way that allowed for easy removal would have potentially introduced non-trivial weight on the segments and altered the finger's dynamics, leading to skewed benchmarking results.

For these reasons, I settled on an alternative approach based on computer vision that would provide more flexibility and avoid reliance on an expensive, immobile, and less user-friendly MoCap setup.

3.3 Tracking Markers

I had access to a simple script from the Soft Robotics Lab that could track specific pixel patterns recorded in a video. The algorithm functions as follows: One manually selects unique geometric pixel patterns that should be followed in the first frame of the video. Then, for each subsequent frame, the algorithm checks whether a similar pattern is near the last position. If a match is found, it correctly designates it as the new position of the tracked element. For my application, I employed round checkerboard targets (see Fig. 16) since they offer robust detection, thanks to their distinct feature points. I could easily print multiple of these markers onto a sheet of paper and glue them onto any object.

Given the simplicity of the code and the ease of attaching the markers to the side of the finger, I wanted to conduct some initial tests with them as a proof of concept. At this stage, I didn't concern myself too much with data synchronization[12]. Consequently, I opted to utilize the Slow-Motion Camera of an iPhone 13 Pro, which records at 240 Hz, without addressing data synchronization at this point.

To derive both the proximal angle (joint 1) and the distal angle (joint 2) of the index finger, I needed to determine the orientation of the three segments A,B & C. Assuming that all markers were located at the same distance from the camera in the z-direction, it sufficed to use the 3D

[12]Data synchronization refers to the process of aligning or coordinating different sets of data in terms of time to facilitate meaningful comparisons and analysis.



Figure 16: Round Checkerboard Target used for Proof of Concept

coordinates in the x and y directions of exactly two markers to find the orientation of each segment as depicted by the vectors $\underline{u}$, $\underline{v}$, $\underline{w}$ in Fig. 17.

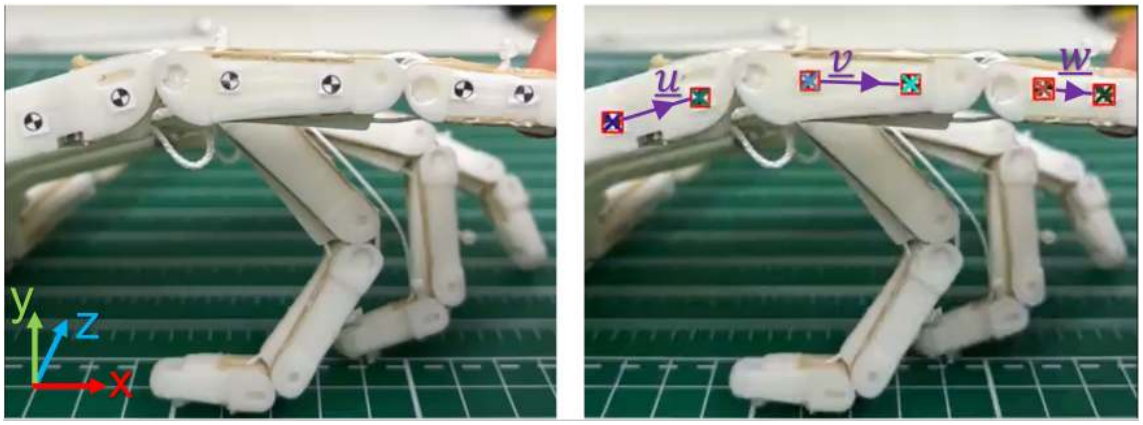


Figure 17: Sideview of the index finger with attached markers. The computer vision algorithm detects the 2D positions of the markers in the xy-plane. These values can be converted into 3D coordinates, enabling the determination of segment orientations represented by $\underline{u}$, $\underline{v}$, $\underline{w}$ and, consequently, the proximal and distal angles of the finger for each frame.

To obtain the precise 3D coordinates of the markers from a 2D plane image, it was essential to determine the camera's intrinsic matrix. This matrix defines the transformation between 3D camera coordinates and 2D homogeneous image coordinates. Using the MATLAB Calibration Tool[13], I successfully determined the intrinsic matrix of the iPhone Camera (Appendix A.3).

The vectors $\underline{u}$, $\underline{v}$, and $\underline{w}$ are derived by subtracting the 3D coordinates of one marker from another along each segment. When assuming that these vectors have no components in the z-direction, calculating both the proximal angle θ_{proximal} and the distal joint angle θ_{distal} becomes straightforward:

$$\theta_{\text{proximal}} = \pi - \arccos\left(\frac{\underline{u} \cdot \underline{v}}{\|\underline{u}\| \cdot \|\underline{v}\|}\right) \tag{1}$$

$$\theta_{\text{distal}} = \pi - \arccos\left(\frac{\underline{v} \cdot \underline{w}}{\|\underline{v}\| \cdot \|\underline{w}\|}\right) \tag{2}$$

Consequently, with this functional system in place, I began to implement logic for data collection and analysis. This endeavor led to significant enhancements to the original marker detection script, including:

[13] https://www.mathworks.com/help/vision/camera-calibration.html, 2023.

- Integration of 3D Reprojection Logic to derive 3D Coordinates.
- Storage of 3D Coordinates in a CSV File.
- Calculation of Joint Angles from the CSV File, together with plotting.
- Implementation of a Moving Average Filter.

Following these code modifications, a series of tests were conducted to validate the accuracy of the 3D Reprojection Logic. I accomplished this by shifting the markers on a grid by a predetermined distance in both the x and y directions, and then comparing these values with the results obtained from the 3D reprojection process. Additionally, I evaluated the accuracy of my joint angles measurements by comparing the angles calculated using formulas (1) and (2) with the actual angles observed in the video.

The proof of concept proved to be a success: I had coded a well-working framework for offline marker position data analysis that I could build upon. An example of the offline angle evaluation results from the proof of concept using the iPhone is demonstrated in Appendix A.1.

However, a significant challenge remained unaddressed in my proof of concept: the critical issue of synchronization.

Within the ROS program controlling the Faive hand, two specific topics that are published (Fig. 18) are of utmost importance:

- **/Faive/commanded_joint_angles (Commanded Angles)**
- **/Faive/joint_angles (EKF Proprioception Angles)**

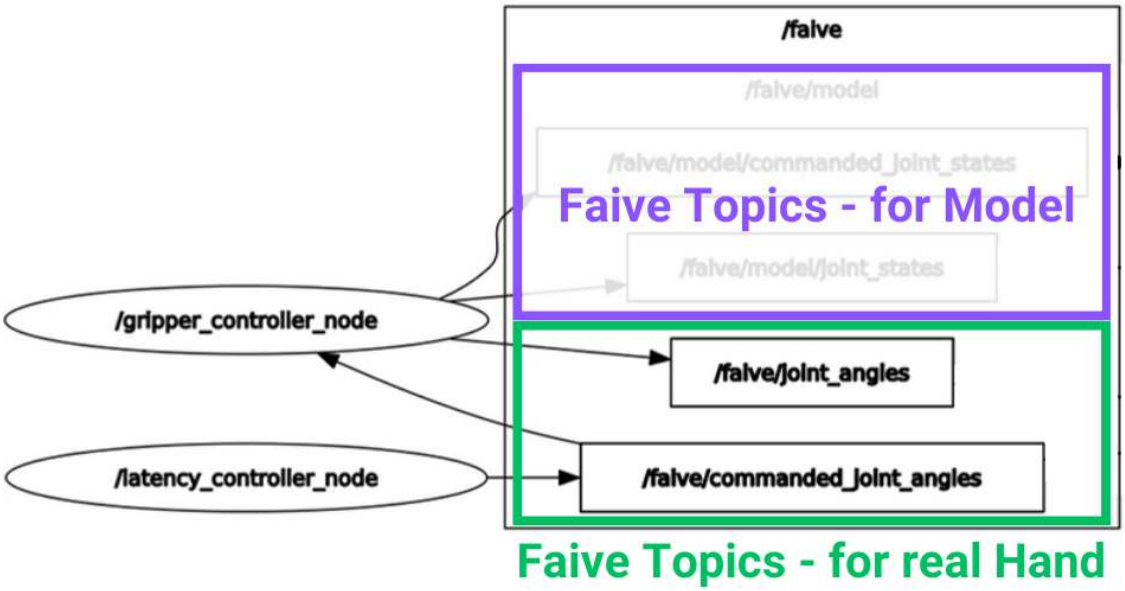


Figure 18: ROS rqt_graph, depicting the publication of Commanded Angles and EKF Proprioception Angles for the physical hand and its simulation model. Only the topics highlighted in green need to be considered as the properties of the real, physical hand, not the simulation model, are benchmarked.

The former topic, Commanded Angles, is responsible for transmitting instructions to the real Faive hand's servos, directing them to rotate the finger joints to specific angles. To evaluate the latency between action command and action execution, a comparison between the Commanded Angles and the Ground Truth Angles from the video recording was essential. Achieving this comparison necessitated millisecond-level synchronization.

On the other hand, the latter topic, EKF Proprioception Angles, represents the angles measured by the servos through proprioception, utilizing an Extended Kalman Filter[14] (EKF). To establish a meaningful comparison between this proprioception data and the ground truth data, synchronization between EKF Angles and Ground Truth Angles was also needed.

Therefore, finding a solution for data synchronization became crucial. However, this endeavor posed a notable challenge, particularly when utilizing an iPhone, as establishing an unrestricted connection between the iPhone's camera and the computer proved to be inherently impossible. Such a connection is essential for precisely aligning the timestamps of the recorded iPhone video with the concurrently gathered data on the computer.

Fortunately, a potential solution emerged: By employing a video recording device compatible with ROS, I could publish the video feed as a ROS topic, enabling synchronization with millisecond-level precision[15]. The critical requirement was that the device had to be capable of recording at a high enough frame rate.

After careful consideration, I opted for an OAK-D S2 device. This device offers two recording options: 60 FPS with autofocus and an RGB feed (RGB Camera), or 120 FPS with no autofocus and a stereo feed (MonoCamera), as seen in Appendix A.2). Initial tests at 60 FPS revealed deficiencies; rapid finger movement resulted in blurriness and marker detection failure. Consequently, I turned to the 120 FPS stereo cameras for a viable solution.

Configuring the ROS system to publish the video feed at 120 Hz instead of only 60 Hz posed a challenging task, requiring thorough research. Fortunately, my familiarity with ROS attributed to my preparatory work facilitated this process, allowing me to program the OAK-D Video Feed Publisher script found in Appendix A.5. Subsequently, I determined the intrinsic matrix of the OAK-D S2 MonoCamera using the ROS camera calibration tool[16] (Appendix A.4).

The final step involved adapting my data analysis code to support rosbags [17] alongside standard video files. After these preparations, I was ready to conduct the first test with synchronized data.

The updated ROS communication structure, now also featuring the video feed from the Oak-D S2, is illustrated in Figure 19.

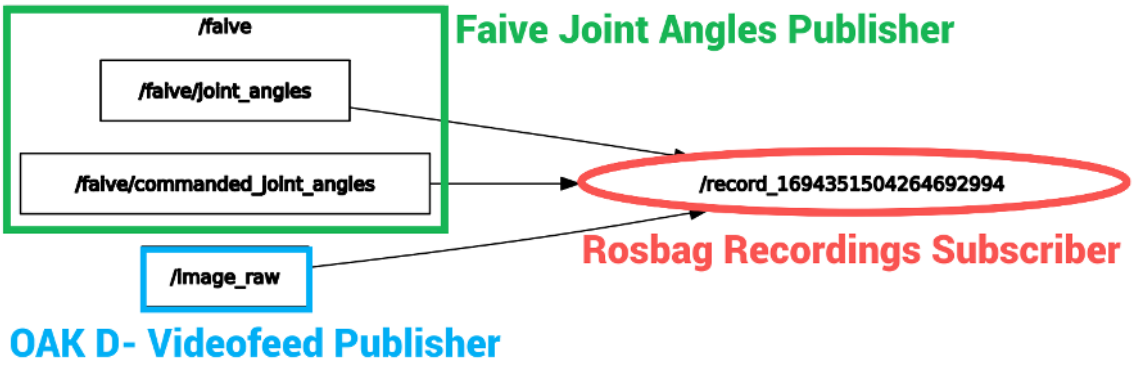


Figure 19: ROS rqt_graph, depicting the publication of Commanded Angles (20 Hz), Proprioception Angles (20 Hz), and, additionally, the live OAK videofeed (120 Hz) to the same rosbag. The Ground Truth Angles can be offline calculated from the videofeed recording and then be compared to the other recorded angles for benchmarking.

During the motion of the index finger and the concurrent camera recording, I captured all the

[14] Nonlinear recursive estimation algorithm for state estimation, particularly in noisy and uncertain environments, commonly used in robotics and control systems.

[15] ROS environments running on modern operating systems typically provide timestamps accurate to within a few microseconds.

[16] http://wiki.ros.org/camera_calibration, 2023.

[17] A rosbag, or .bag file, is a binary file format used in the Robot Operating System (ROS) to store various types of data, such as messages, topics, and time-stamped information, facilitating efficient and synchronized data storage and playback.

messages from each topic into a single rosbag, which included corresponding timestamps for each message. The images extracted from the recorded video within the rosbag were then processed by my existing data analysis framework, originally designed for the iPhone. This allowed the calculation and storage of the Ground Truth Angles for each frame in a CSV file. Subsequently, Commanded Angles, EKF Angles, and Ground Truth Angles could all be plotted together, theoretically providing the required raw data for the benchmarking process.

However, a significant challenge arose due to the limitations of the OAK-D S2 MonoCamera. With no autofocus and a minimum focal length of 19.6 cm, the markers appeared blurry and smaller compared to the high-resolution autofocus images obtained from the iPhone (Fig. 20). This resulted in the failure of the marker detection code that I had used as a baseline. When the hand was not moving, the static position measurement noise was very similar for both cameras (± 1mm), as seen in Fig. 21. But when the hand was moving too fast, a very critical issue emerged.

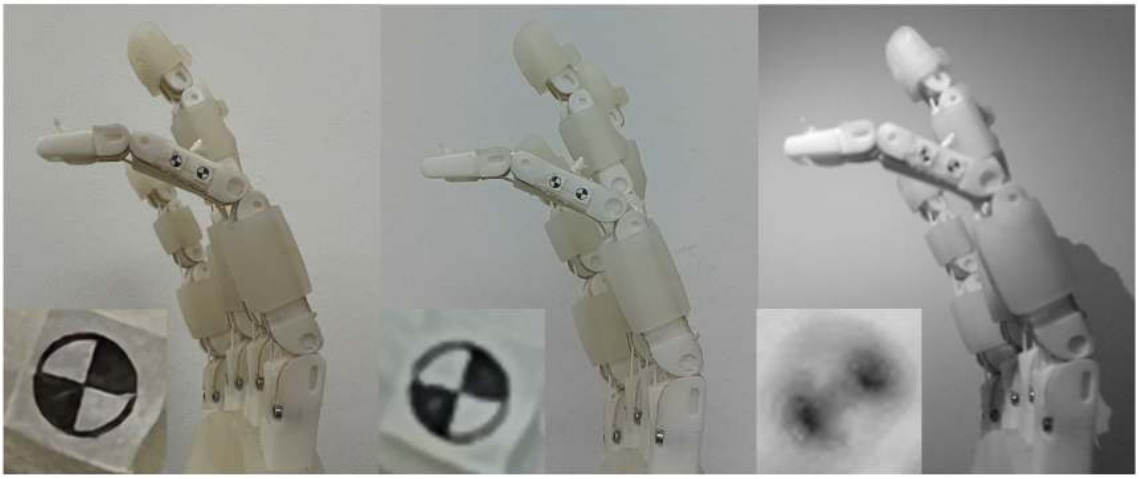


Figure 20: Comparison of video resolutions at a 10 cm depth: iPhone 13 Slowmotion 240 FPS Camera (left) vs. OAK D- S2 RGB 60 FPS Camera (center) vs. OAK D- S2 120 FPS MonoCamera (right). The markers in the right image are more challenging to detect than those in the left and central image for the algorithm due to blurriness especially at high moving speeds.

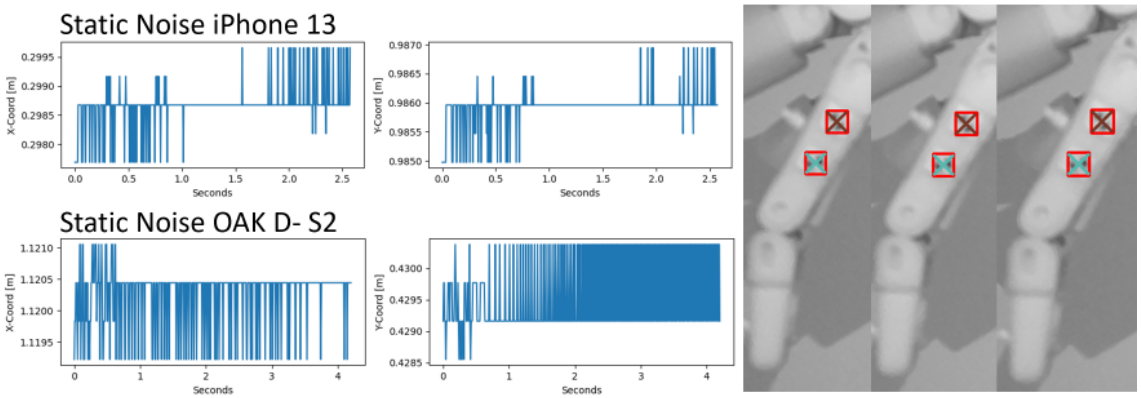


Figure 21: Comparing Static Noise: When markers remain still, iPhone and OAK-D S2 exhibit similar noise.

The combination of image blurriness and increased motion blur during rapid movements often resulted in one or more markers being completely unrecognized, leading to an immediate loss of marker position data. This limitation was further compounded by the algorithm's inability to recover marker positions once lost in a frame, which posed a substantial dilemma (Fig.22). Reverting to the 60 FPS autofocus RGB camera for higher resolution was not a feasible alternative, as even 120 FPS was on the lower end of the required frame rate for obtaining clear, unblurred

video frames during fast movements. Consequently, addressing this intricate and significant issue became imperative.

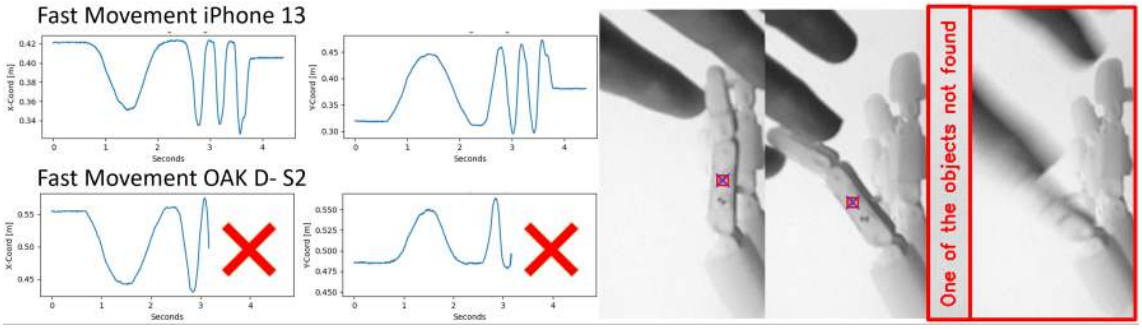


Figure 22: Comparing Dynamic Position Tracking: When markers move quickly, the difference between high-resolution 240 Hz and low-resolution 120 Hz videofeeds becomes evident. On the iPhone, the markers can be tracked at arbitrarily high displacement speeds. However, for the OAK D-MonoCamera, as seen on the right, the algorithm loses track of the marker positions at high speeds due to motion blur, making it impossible to recover them. This unrecoverability is a significant problem considering that the Faive hand's finger need to move very rapidly for the benchmarking experiments.

3.4 Tracking Apriltags

One solution I considered was simply increasing the size of the markers. However, I recognized that this approach was not a robust long-term solution. I needed a more reliable strategy—one that would allow the code to relocate the marker positions even after they were lost for multiple frames. During my research for potential solutions, I came across apriltags. Apriltags are square-shaped markers featuring a unique and intricate pattern reminiscent of QR codes (as illustrated in Fig. 23). Each individual pattern is associated with a distinct and exclusive identification number, allowing for precise differentiation between various apriltags. These markers are designed to facilitate advanced tracking and identification techniques, making them an ideal choice for applications requiring accurate and automatic marker detection and localization. Most importantly, it allows for automatic tracking without the need for manual selection. Consequently, even if an apriltag is undetected for multiple frames due to blur, the algorithm automatically rediscovers it as soon as the blur subsides.



Figure 23: Visualization of multiple apriltag formats. For my application, I used Tag36h11. All formats showcase a unique square pattern designed for precise position and orientation detection.

In addition to their automatic tracking capability, apriltags provide a significant advantage—they not only enable tracking of their 3D position but also their orientation as shown in Fig, 24. This allowed me to replace the use of two separate round markers to determine the orientation of each finger segment with just one apriltag per segment. This design change allowed for a substantial increase in the size of the apriltags, further enhancing tracking robustness. While the old markers exhibited positional tracking noise of about ± 2mm when using the OAK-D S2 MonoCamera due to blurry video feeds, the introduction of apriltags reduced this value to less than ± 1mm.

This marked a substantial improvement. To assess how this enhancement would impact joint angle measurements, I adapted my ground truth analysis framework to incorporate apriltag detection

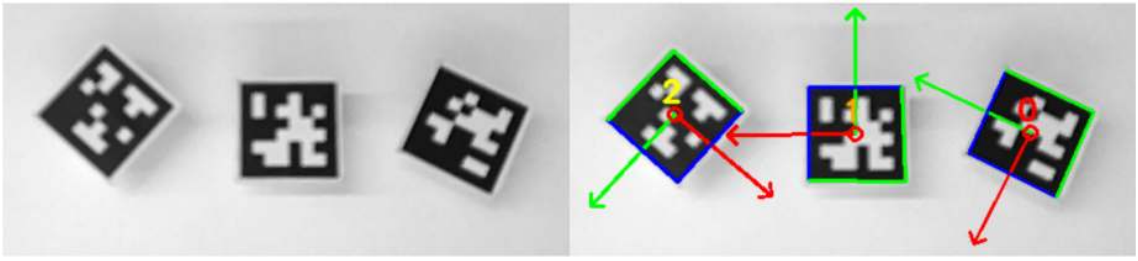


Figure 24: Despite the OAK MonoCamera's limited resolution, the apriltag algorithm accurately determindes the identification numbers (0, 1, and 2), the position of the center, and the 3D orientation in space of each apriltag. The tag orientation is defined by two perpendicular vectors, and the center position is represented by 3D coordinates relative to the camera's origin frame.

using the apriltags python library. I also needed a method to attach the Apriltags to the fingers in a manner that ensured perfect alignment with their respective segments. The rationale behind this alignment was to ensure that when the finger was fully extended, the Ground Truth Angles I would measure would be precisely zero degrees, consistent with the simulation model and proprioception implementation of the hand (see Fig. 26 & 27). By achieving accurate alignment of the Apriltags with each segment, I could establish a meaningful comparison between Ground Truth and Proprioception Angles, thus facilitating precise benchmarking of proprioception accuracy. Consequently, I set out to design lightweight apriltag holders in CAD that could be easily attached and detached from the finger segments, aligning the Apriltags to correspond with the definition of Ground Truth, Commanded, and EKF Proprioception Angles.

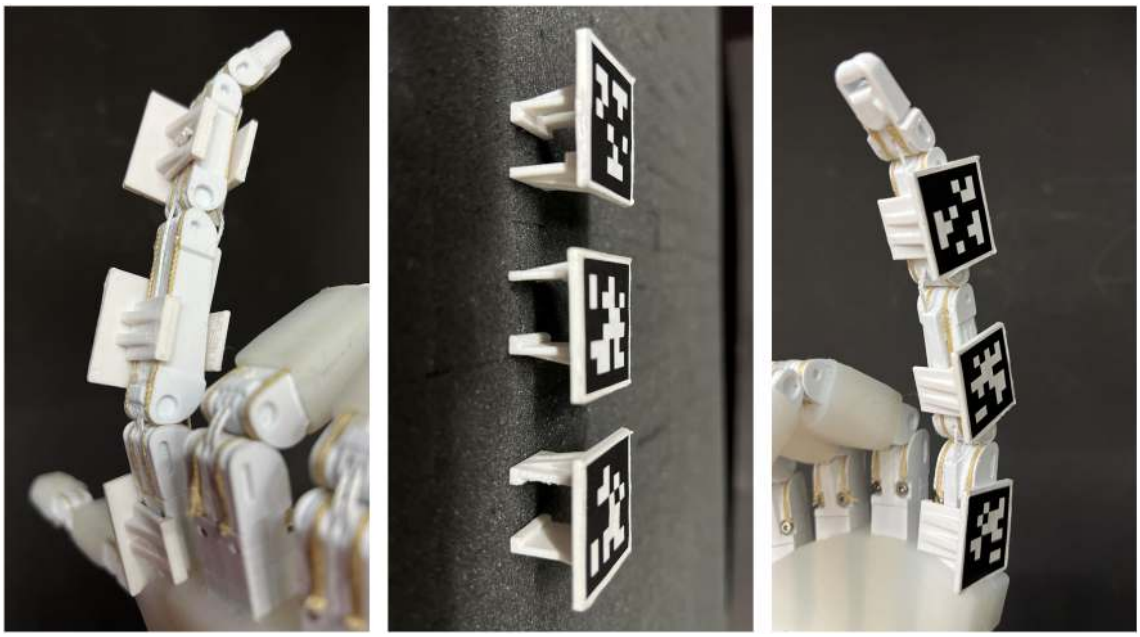


Figure 25: Depiction of three distinct Apriltag holder designs showcasing variations tailored to the unique geometric properties of each finger segment. These designs prioritize user-friendly attachment and detachment, maintain a flat surface, ensure precise alignment with the finger, offer long-lasting durability and stability, all while being lightweight and easy to print.

Given that each finger segment possessed distinct geometrical properties, I had to design three unique holders (see Fig. 25). To enhance flexibility and ease of attachment and detachment, I used PETG as 3D-printing material. I selected three different Tag36h11 patterns, each corresponding to a finger segment. This selection facilitated differentiation between the segments in the code and automated the entire data analysis process.

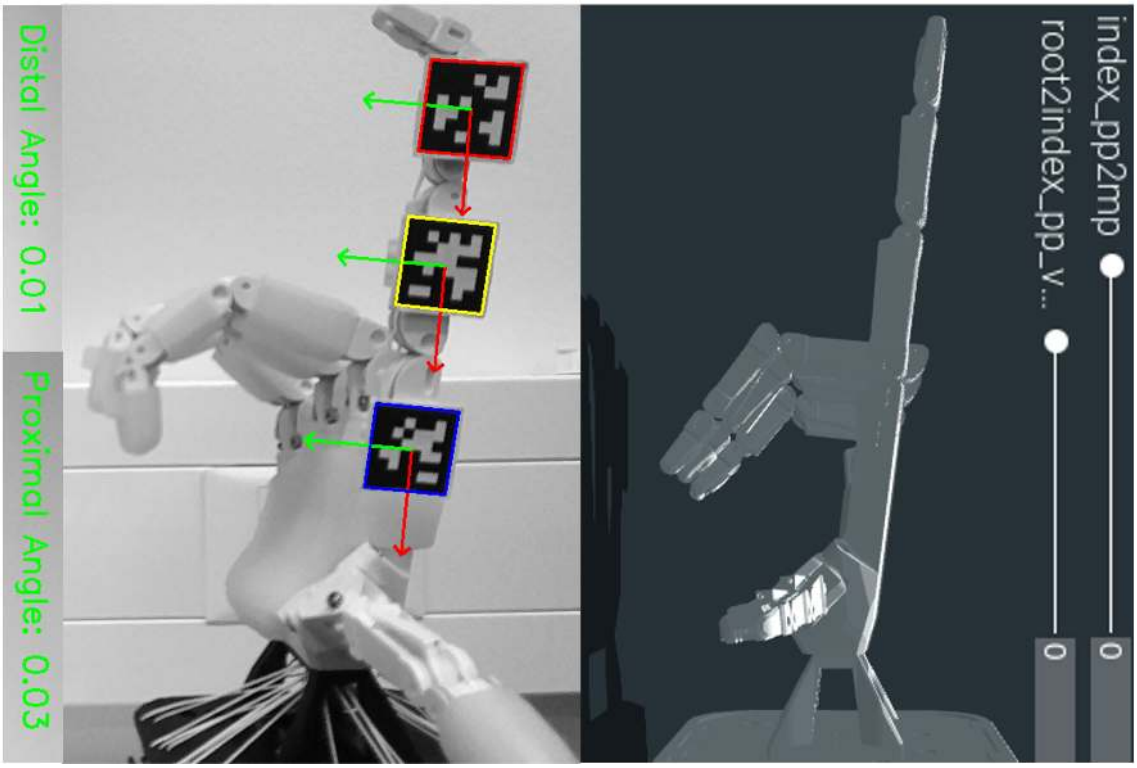


Figure 26: Depiction of the real hand (left) and the URDF hand model (right): Proximal angle (root2index_pp) and distal angle (index_pp2mp) are both precisely zero when the index finger is fully extended. Alignment with perfect apriltag positioning was essential to match real hand joint angle measurements closely to zero degrees during full extension, ensuring consistency in angle definitions between Ground Truth and Proprioception/Commands data.

With this configuration in place, I was finally prepared to evaluate the performance of the Apriltag detection system on the Faive hand. When the index finger remained stationary, experiencing no movement, the noise level was a mere $\pm$ 0.3 degrees, as illustrated in Fig. 28. This accuracy level was more than sufficient for my proprioception benchmarking, especially considering that the inherent proprioception accuracy itself did not extend to the first significant digit.

When manually flexing and extending the proximal joint angle over a period of time, the observed ground truth data derived from the tracked apriltags was subsequently virtually noise-free, as shown in Fig 29. It became evident that the various moving average filters I had implemented to refine the previous marker data were rendered redundant due to the precise tracking of apriltag orientation vectors. Even during rapid finger movements or instances of tag occlusion from the camera's field of vision, the tracking algorithm managed to reestablish tag recognition once the motion slowed or the occlusion ceased.

I also incorporated logic to calculate the angle not solely within the x-y plane, but rather within the rotational plane defined by the orientation vector of the apriltag on the proximal segment. However, this adjustment did not yield improvements in my calculations. As a result, I maintained the assumption that the markers were approximately perpendicular to the camera. To ensure this, I developed a specialized script that displayed the live video feed from the OAK-D S2 camera and provided apriltag perpendicularity and alignment information. This script served as a tool to confirm zero-degree alignment of markers when the hand was fully extended (as has been seen in Fig. 26) and to maintain their perpendicularity to the camera within a 5-degree margin, which is illustrated in Fig. 30. Prior to conducting any ground truth measurements, I could consistently verify accurate marker mounting and camera alignment with the finger using this script, which was important for meaningful ground truth data.

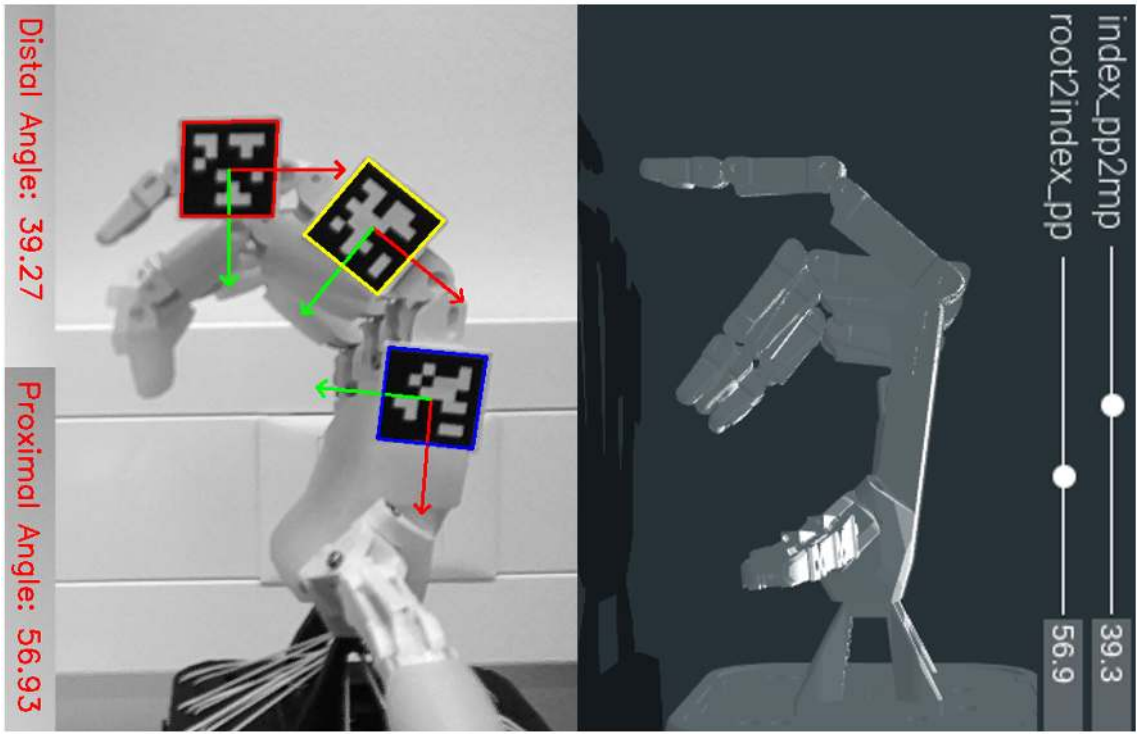


Figure 27: After accurately aligning the apriltags with the finger segments, tests were conducted to assess consistency during flexion. The real index finger (left) was moved to various positions, and the measured angles were compared to those of the URDF model (right).



Figure 30: In each frame of the live OAK videofeed, apriltags' perpendicularity to the camera was checked within a 5-degree error margin in both x and y directions by evaluating orientation vector cross-products. That way, it could be evaluated whether the hand was misaligned (left) or not (right). This step was critical as joint angle calculations were performed in a 2D plane, assuming a constant marker distance from the camera. Ensuring proper apriltag perpendicularity before the benchmark experiments mitigated errors caused by camera and hand misalignment.

With this, my initial objective was accomplished: I had successfully developed an accessible framework capable of recording synchronized ground truth data for the joint angles of the Faive index finger. The stage was now set to employ this framework and commence the crucial benchmarking of both latency and proprioception accuracy for the Faive hand.

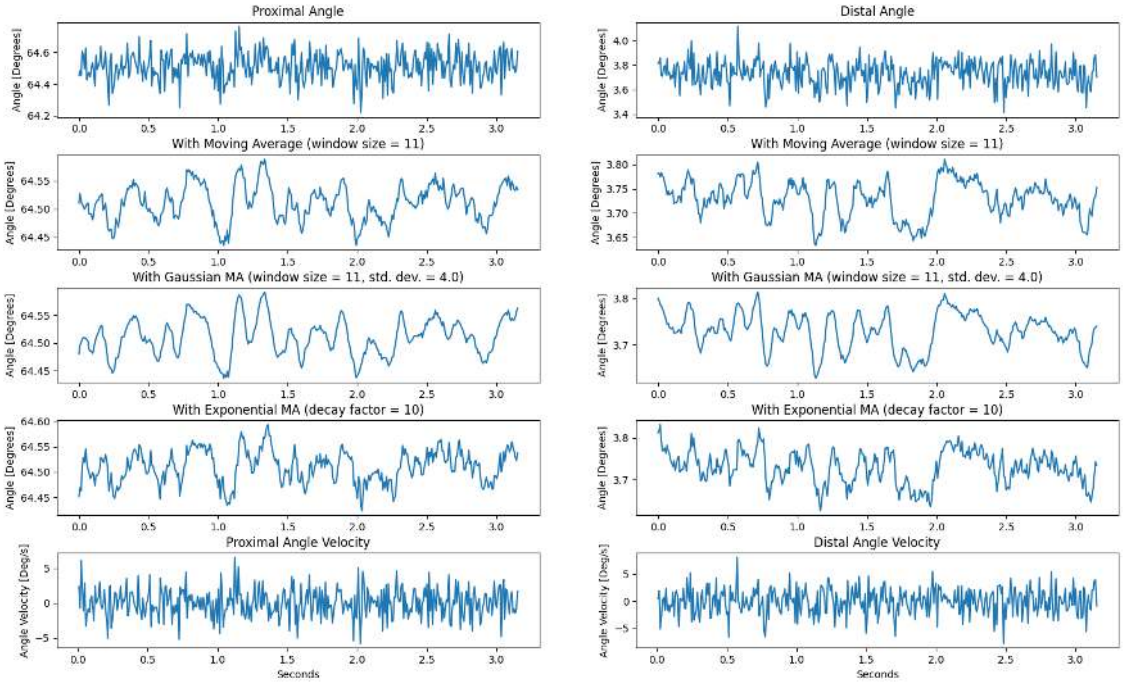


Figure 28: Ground truth data obtained when the finger is stationary. The plot illustrates that the static noise remains within a narrow range of approximately $\pm$ 0.3 degrees, a level of accuracy surpassing that achieved with the round markers substantially. The application of moving averages further reduces the impact of static noise.

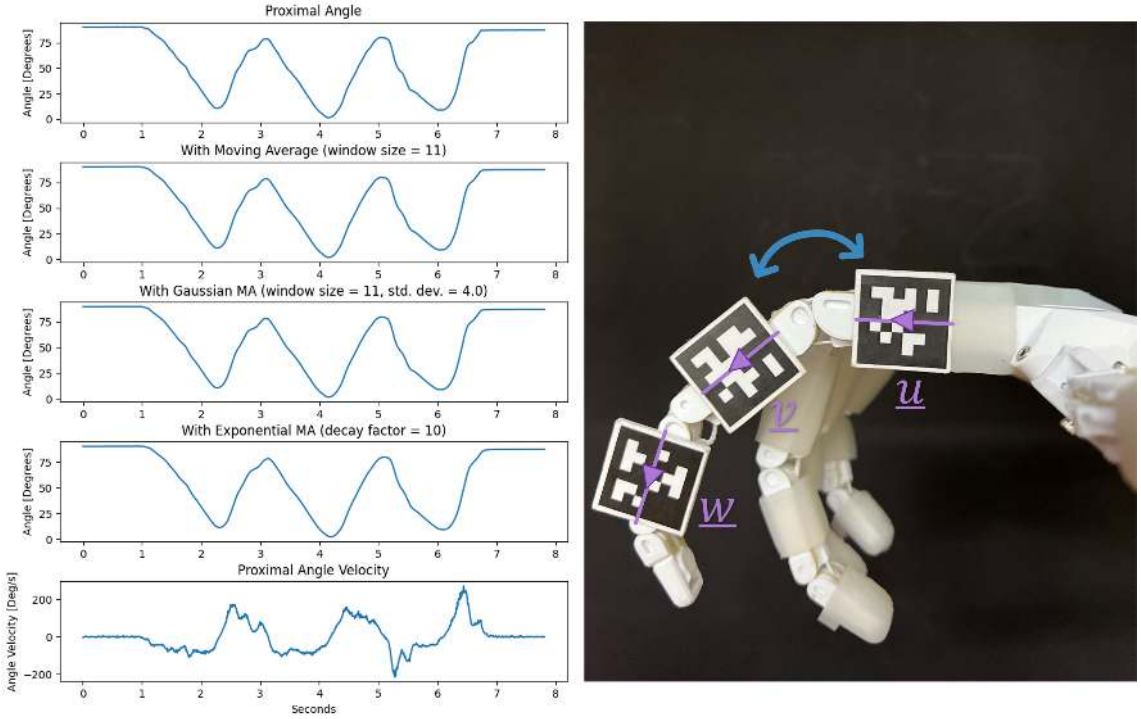


Figure 29: Ground truth data during manual flexion and extension of the proximal joint by hand. The plot demonstrates a negligible level of measurement noise in comparison to the actual angle change, rendering the noise nearly imperceptible. Notably, the application of moving averages has limited effect here due to the high signal-to-noise ratio. The calculated velocity appears more noisy, as it is obtained by taking the temporal derivative of the data points. The joint angles are calculated the same way as in formulas (1) and (2) using the orientation vectors $\underline{u}$, $\underline{v}$, $\underline{w}$.

3.5 Benchmarking Latency

To assess the system's latency, or in other words the time disparity between Commanded Angles and Ground Truth Angles, I ran a straightforward open-loop control program that does the following: As illustrated in the video frames in Fig. 31, the index finger periodically flexes and extends its proximal joint to varying angle values.

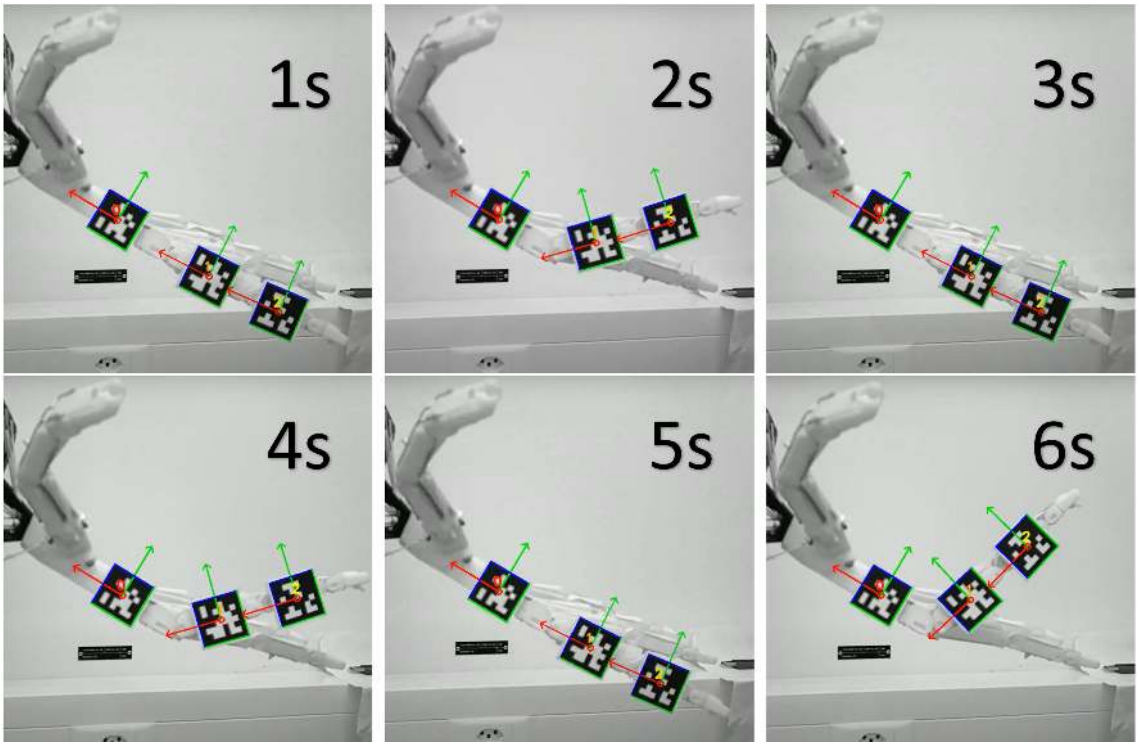


Figure 31: Cyclic movement of the index finger, alternating between flexing and extending its proximal or distal angle at a frequency of approximately 0.5 Hz. The finger begins in an extended position, then flexes to a specific commanded angle, maintaining it for approximately 1 second. It subsequently extends back to the initial position for an additional 1 second. This cyclic pattern repeats, with the angle varying across trials to assess latency changes at different angle values.

The first plot obtained through my framework is shown in Fig. 32

The plot reveals a noticeable time difference between the Commanded Angles (shown in red) and the actual angles or Ground Truth Angles (displayed in blue). Additionally, an immediate observation is that the Proprioception (shown in orange) and Ground Truth angles did not align well in the initial experiments. My initial priority was to determine the latency. To achieve this, I implemented a custom algorithm[18] that identified the edges of all square signals depicted in Fig. 32 as vertical lines. This methodology enabled me to extract precise timestamps from two paired lines, facilitating the calculation of latency.

Interestingly, one can observe that Proprioception Angles exhibited an earlier rise and fall compared to the Ground Truth Angles—a seemingly illogical behavior. Through experimentation, I discovered that the reason behind this phenomenon was tendon slack. While the hand's servos could initiate movement, the finger itself would commence motion only after the tendon had been fully stretched. This issue was resolved by tightening the finger's tendons, as demonstrated in Fig. 33.

With these enhancements in place, I proceeded to perform a quantitative assessment of latency

[18]The algorithm employs a sliding window approach, computing the variation within the window. When the calculated variation exceeds a defined threshold and the window's length satisfies a defined minimum criteria, a stable window is identified. This procedure iterates to detect all stable windows, allowing for the determination of square signal edge timings.

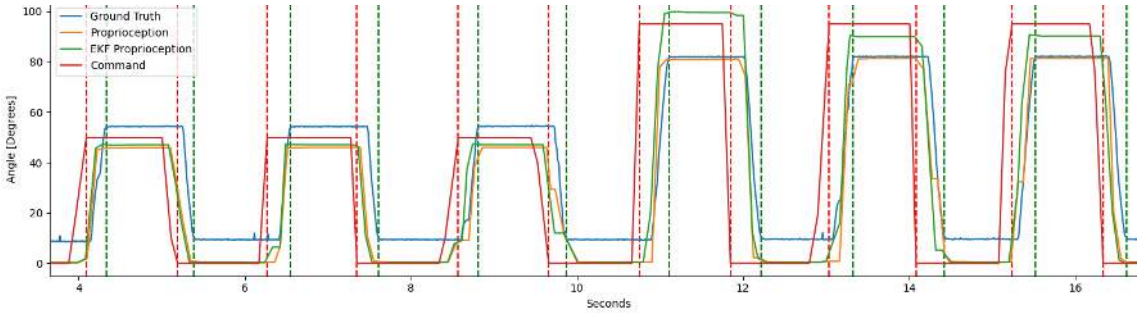


Figure 32: This plot displays synchronized data for Commanded Angles (Command), EKF angles (Proprioception), and Ground Truth Angles over time. Notably, there is a visible time disparity between the Command and Ground Truth signals, indicated by vertical lines at their edges. Additionally, imperfect alignment can be observed between Proprioception and Ground Truth. Please note that in all subsequent plots, the 'Proprioception' graph is omitted, and the 'EKF Proprioception' graph is renamed to 'Proprioception'.

Latency Values - Initial Experiment		
	Proximal Angle	Distal Angle
Small Angle - Flexion	0.2817 ± 0.0933 s	0.2225 ± 0.0507 s
Small Angle - Extension	0.2520 ± 0.1203 s	0.2210 ± 0.0427 s
Medium Angle - Flexion	0.3232 ± 0.0570 s	0.3533 ± 0.0942 s
Medium Angle - Extension	0.2967 ± 0.0362 s	0.3427 ± 0.1034 s
Large Angle - Flexion	0.3613 ± 0.0484 s	0.3429 ± 0.0498 s
Large Angle - Extension	0.3658 ± 0.0832 s	0.3583 ± 0.0806 s

Table 1: Average Latency & Latency Standard Deviation for different Finger Movements (3 full Cycles)

for varying degrees of finger movement: small, medium, and large angles. Specifically, I classified small angles as those below 30 degrees, medium angles within the range of 30 to 60 degrees, and large angles spanning from 60 to the theoretically maximum 90 degrees of flexion.

Executing the test cycle illustrated in Fig. 33 three times for both proximal and distal angles, I computed the average latency values. It's important to highlight that I differentiated between instances where the finger flexed from a fully extended position to a specific angle and instances where it extended back from an angle to the fully stretched position, due to the distinct movement patterns of the tendons in these two actions. The resulting average latency values, along with their corresponding standard deviations, are summarized in Table 1.

Table 1 clearly highlights the correlation between increased Commanded Angle amplitudes and the corresponding time taken by the finger to reach those angles, leading to higher latency. This observation implied that if I intended to train the Faive hand for very fast behavior, which involves substantial angle changes in a short period of time, the hand would have to deal with increased latency. Moreover, it's noteworthy that differentiating between flexion and extension movements had minimal impact on the timing, indicating that both types of movements could be treated similarly with regards to latency analysis. However, despite these insightful observations, the overall outcomes remained unsatisfactory.

The latency not only exceeded 0.35 seconds for large angle changes but also exhibited a considerable standard deviation, indicating a significant degree of variability in latency across the experiments. The notable variability in latency was a concerning revelation as it made the implementation of latency in the Faive simulation model virtually unfeasible.

Furthermore, it was concerning that as shown in Fig. 33, at the third square signal of the third wave in the upper graph, instances could arise where the latency even exceeded 0.5 seconds. Such high latency values are highly impractical for achieving robust, swift dexterous manipulation. This observation compelled me to undertake a thorough investigation into the root cause of this

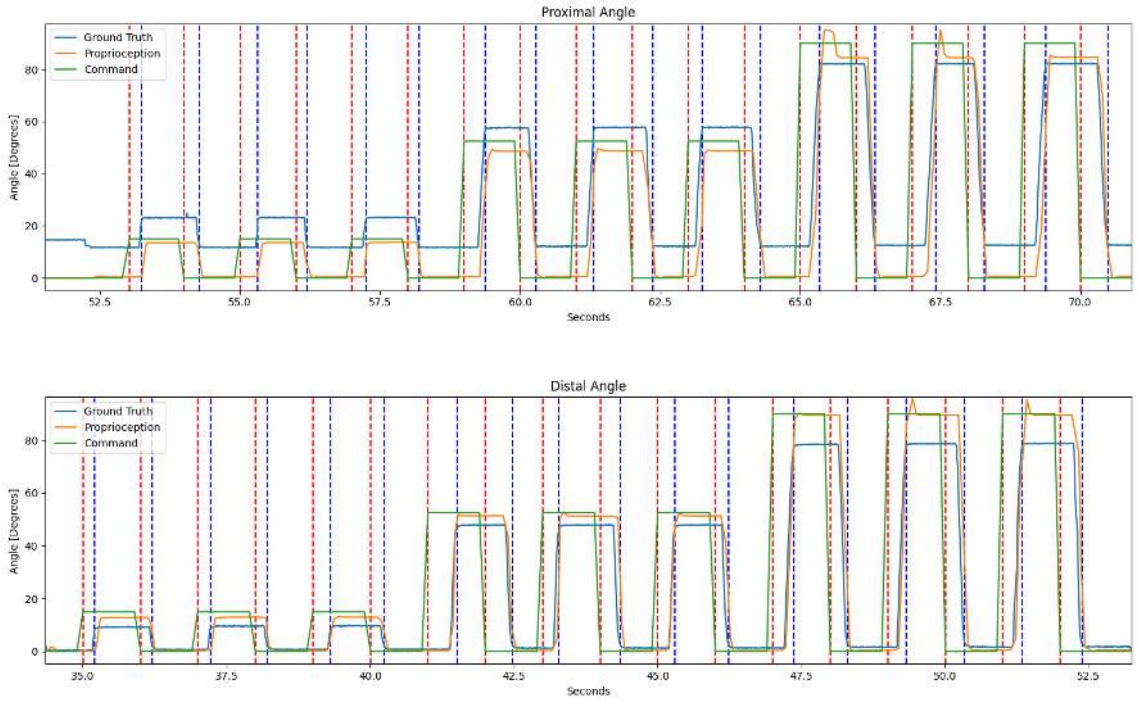


Figure 33: Plot illustrating a complete latency test cycle for both the proximal and distal joints, consisting of three square signals for three distinct angles each.

Latency Values - without Publishing EKF Angles		
	Proximal Angle	Distal Angle
Small Angle - Flexion	0.1939 ± 0.0058 s	0.1873 ± 0.0103 s
Small Angle - Extension	0.1983 ± 0.0058 s	0.1919 ± 0.0044 s
Medium Angle - Flexion	0.2703 ± 0.0056 s	0.2552 ± 0.0119 s
Medium Angle - Extension	0.2700 ± 0.0053 s	0.2610 ± 0.0050 s
Large Angle - Flexion	0.3265 ± 0.0103 s	0.2996 ± 0.0069 s
Large Angle - Extension	0.3201 ± 0.0051 s	0.3178 ± 0.0135 s

Table 2: Average Latency & Latency Standard Deviation for different Finger Movements (3 full cycles)

unpredictability. To gain deeper insights, I performed tests under conditions where only Commanded Angles and the OAK Videofeed were published, excluding Proprioception Measurements. Intriguingly, this modification yielded a reduction in both latency and latency standard deviation (Table 2), signaling a potential link between the presence of Proprioception Measurements and the observed latency inconsistencies.

By delving into the issue, I was able to identify the root cause of the problem within the EKF calculation code used for Proprioception. Through thorough analysis, it became apparent that a significant portion of the latency, approximately 25 percent, could be attributed to a matrix inversion operation within the script. This operation's execution time varied depending on the specific matrix being inverted, accounting for the observed wide range of latency values.

Listing 1: Python Code inverting Matrix S for EKF Calculation

```
# Inversion of matrix S
numpy.linalg.inv(S)
vs.
# Leverage fact that S is sparse for more efficient computation
scipy.splinalg.inv(scipy.sparse.csc_matrix(S))
```

Latency Values - with new Inversion Method		
	Proximal Angle	Distal Angle
Small Angle - Flexion	0.1070 ± 0.0053 s	0.1027 ± 0.0062 s
Small Angle - Extension	0.1127 ± 0.0067 s	0.1118 ± 0.0044 s
Medium Angle - Flexion	0.1896 ± 0.0078 s	0.1698 ± 0.0057 s
Medium Angle - Extension	0.1823 ± 0.0057 s	0.1771 ± 0.0050 s
Large Angle - Flexion	0.2487 ± 0.0072 s	0.2235 ± 0.0072 s
Large Angle - Extension	0.2413 ± 0.0053 s	0.2325 ± 0.0135 s

Table 3: Average Latency & Latency Standard Deviation for different Finger Movements (2 full cycles, see Fig. 34)

Fortunately, a solution presented itself. The matrix in question turned out to be sparse, containing numerous zero values. Leveraging this characteristic, I transformed the S matrix into a Compressed Sparse Column (CSC) format before performing the inversion, as demonstrated in the code provided in Listing 1. This optimization substantially improved the efficiency of the computation, reducing the average latency introduced by the inversion from 0.05 seconds to a mere 0.005 seconds. The impact of this change was strikingly evident, as reflected by the CPU usage on my laptop, which surged from 80% to 800%, underscoring the significance of this matrix inversion in driving the increased latency.

Following this relatively minor code adjustment, the latency profile underwent a remarkable transformation, as evidenced by the experiment depicted in Fig. 34 and Table 3.

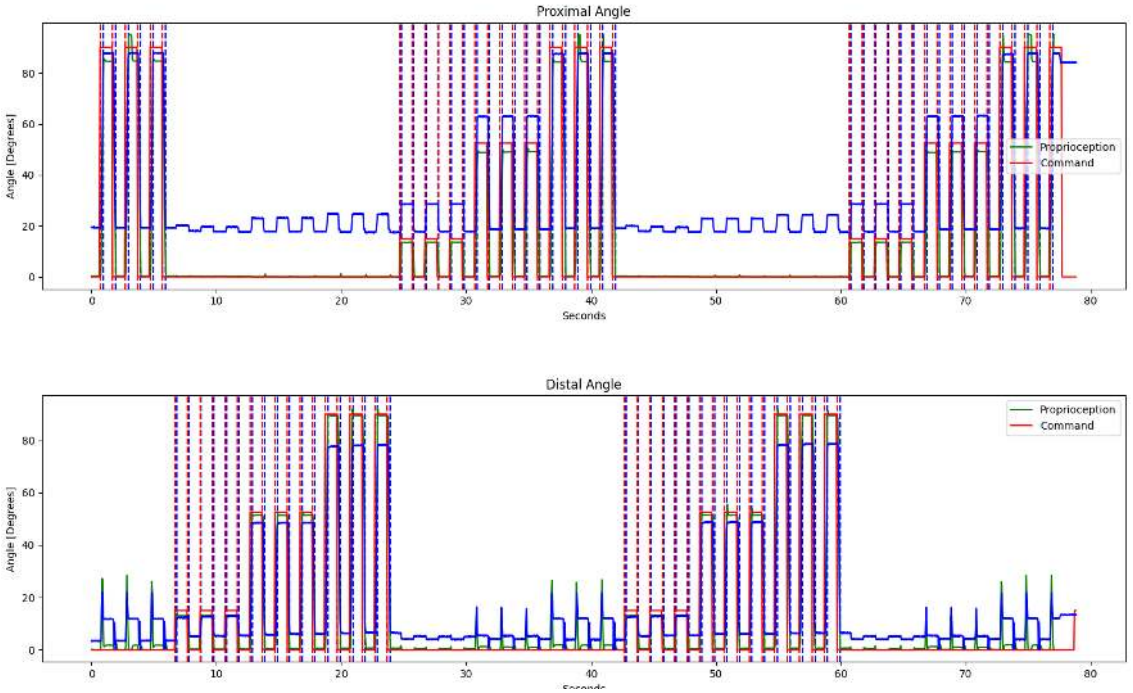
This improvement was truly remarkable, resulting in a substantial reduction in both latency and latency standard deviation across all movement categories:

- **Small Angles Latency:** 0.2443 s → 0.1086 s
- **Small Angles Latency Standard Deviation:** 0.0768 s → 0.0057 s
- **Medium Angles Latency:** 0.3290 s → 0.1797 s
- **Medium Angles Latency Standard Deviation:** 0.0727 s → 0.0061 s
- **Large Angles Latency:** 0.3570 s → 0.2365 s
- **Large Angles Latency Standard Deviation:** 0.0655 s → 0.0083 s

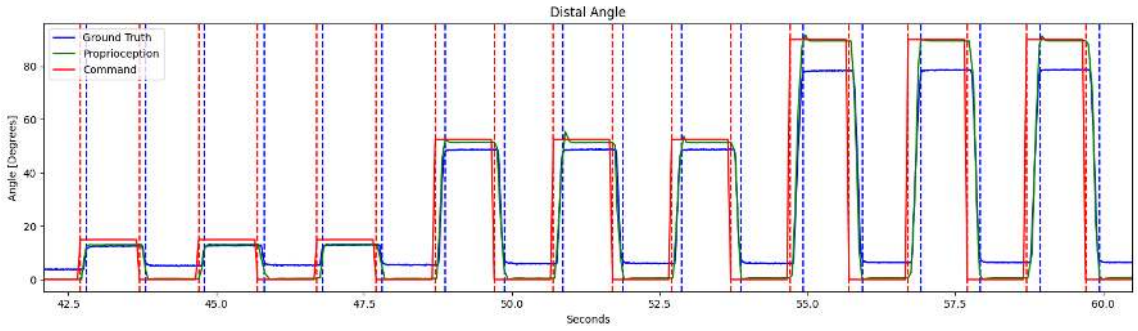
It's worth noting that the natural signal transmission time from the human brain to the hand is estimated to be at around 150 to 220 milliseconds[19]. By successfully benchmarking and as a result also reducing latency, I had achieved human-level, if not surpassing, performance in signal latency. This achievement was promising, as it indicated that latency was unlikely to become a major constraint in my application.

In a final adjustment, I aimed to enhance the hand's agility and responsiveness, especially in scenarios requiring closed-loop control. To achieve this, I increased the publishing rate of the Commanded Angles from 20 Hz to 100 Hz (Fig. 35). By elevating the publishing rate, the hand could more swiftly react to real-time commands and execute complex movements.

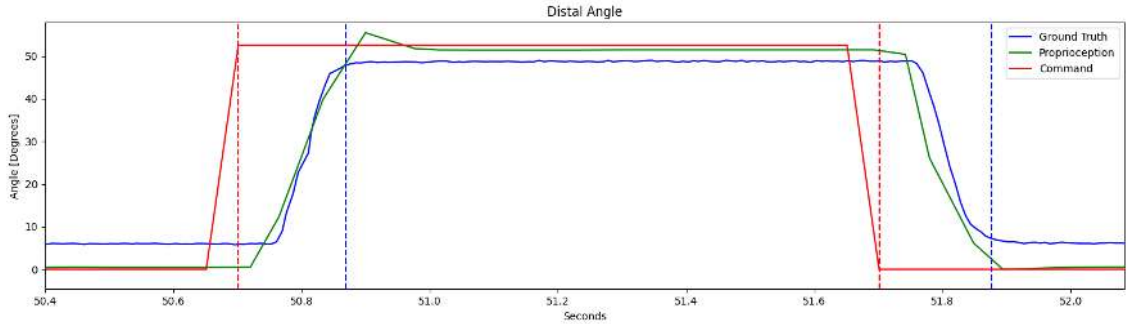
[19] https://quick-advices.com/how-long-does-it-take-for-a-nerve-signal-to-travel/, 2023.



(a) Plot illustrating the complete experimental setup with the newly implemented inversion method, incorporating two full cycles.

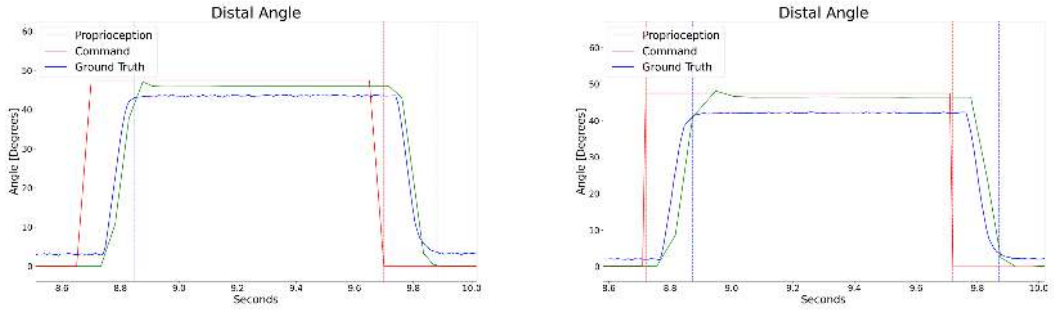


(b) Focused view of a cycle for the distal angle. The plot showcases the increasing separation between vertical pairs as the angle amplitude grows, while maintaining consistent separation for equal amplitudes. This stability is a marked improvement in comparison to the fluctuation seen in Fig. 33



(c) Detailed examination of a single square signal, providing in-depth insight into ground truth data noise, latency, and proprioception accuracy during a specific flexion-extension movement.

Figure 34: Experimental results after new matrix inversion implementation depicted through a series of plots, each offering varying levels of detail and focus.



(a) (b)

Figure 35: Comparison of Square Signals at Different Commanded Angle Publishing Rates: (a) 20 Hz, (b) 100 Hz. The increased Publishing Rate enhances the hand's responsiveness in closed-loop scenarios, enabling quicker reactions and facilitating more intricate movements.

3.6 Benchmarking Proprioception Accuracy

Having successfully completed the latency benchmarking, it was now time to turn my attention to benchmarking the proprioception accuracy of the Faive hand. As evident from the figures in the previous chapter, a significant disparity between the amplitude of the EKF Proprioception Angles and the Ground Truth Angles was almost always observable. Identifying the underlying cause was not a trivial task, but I managed to narrow it down to two primary factors:

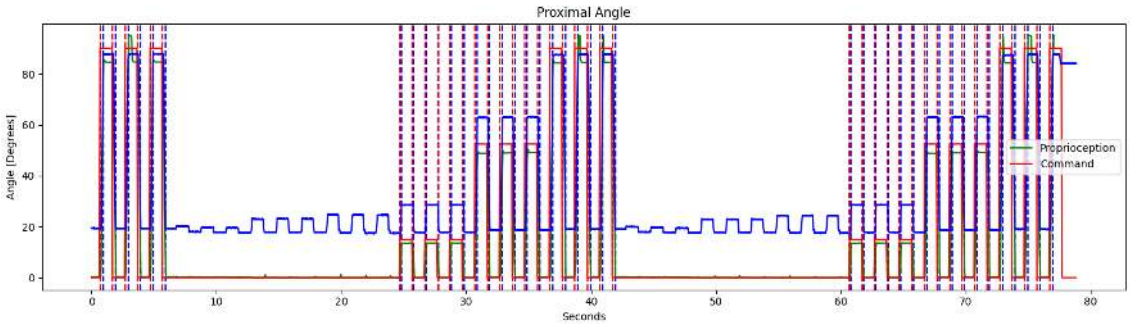
- **Calibration Inaccuracy:** The calibration process for the Faive hand proved to be not very precise[20]. Although in theory, the calibration device aimed to align the fingers perfectly at zero-degree angles in the extended state, practical evaluations using my real-time joint angle assessment script revealed consistent alignment discrepancies of up to 10 degrees in the finger segments, even immediately after calibration. This discrepancy arises because the calibration device does not fully extend the fingers, and tendon tightening during calibration can be inconsistent. This highlighted the need to reevaluate the calibration technique to ensure accurate alignment between the real hand's extended state and the assumed extended state in the simulation model.
- **Accumulation of Slack:** Over prolonged usage, slack gradually built up within the hand's mechanical components. This slack contributed to a growing disparity between proprioception angles derived from servo states and the actual physical angles of the hand.

These observations were really intriguing. Through manual calibration of the index finger, with a specific emphasis on achieving full extension rather than relying on the standard calibration device, I effectively minimized the signal discrepancy by a substantial margin. This improvement is clearly demonstrated in Fig. 36.

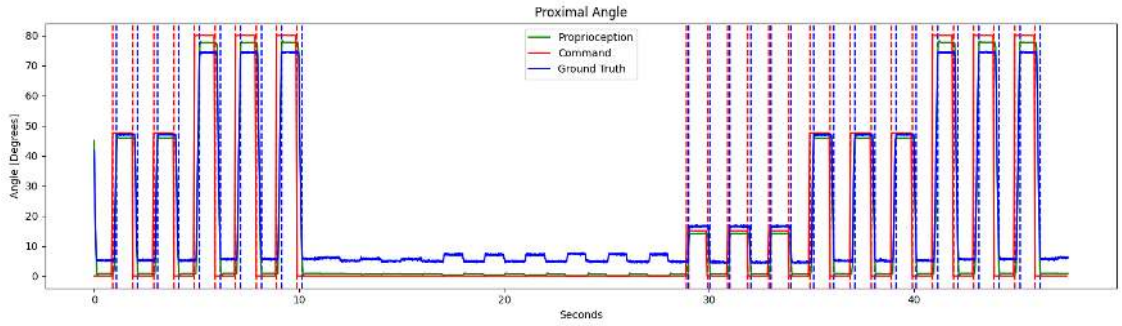
However, my efforts to further minimize the angle discrepancy after calibration proved to be limited. Therefore, I proceeded to conduct a quantitative analysis of proprioception accuracy. For that, I conducted an experiment where the index finger executed a sine wave motion, yielding the graphs shown in Fig. 37. Then I simply plotted the EKF Proprioception Angles datapoints over the Ground Truth Angles datapoints and obtained the insightful plots shown in Fig. 38.

As anticipated due to the tendon-based nature of the Faive hand, I observed a subtle hysteresis. This hysteresis indicates a slight directional variance between the two angles during finger flexion and extension. Employing linear regression, I fitted linear functions to the data points that depicted the relationship between Proprioception and Ground Truth for both positive and negative angle changes. Intriguingly, both regressions exhibited a comparable slope of approximately **1.07**, along with a minor angle offset of around **0.6** degrees. This offset, surprisingly small, implies that the

[20]The calibration process in the Faive hand involves the use of a calibration device, resembling a rigid glove, placed on the fingers. This device helps in achieving partial finger extension while the servos adjust to ensure adequate tendon tension.



(a)



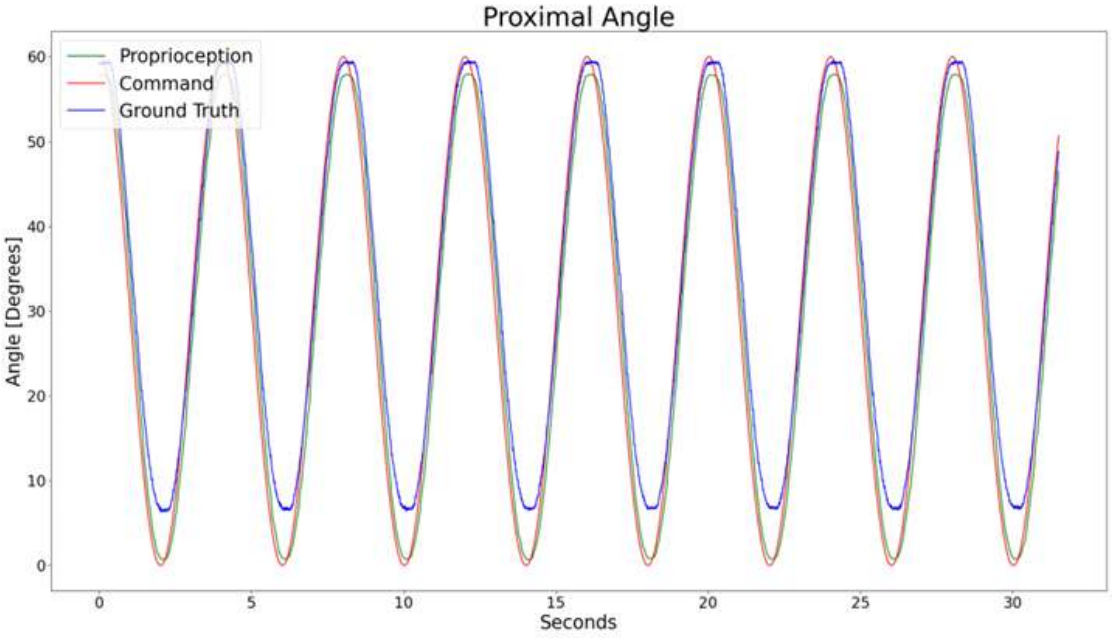
(b)

Figure 36: Comparison of Proprioception Inaccuracy: (a) After calibration using the standard device, (b) After manual calibration with ensured full finger extension.

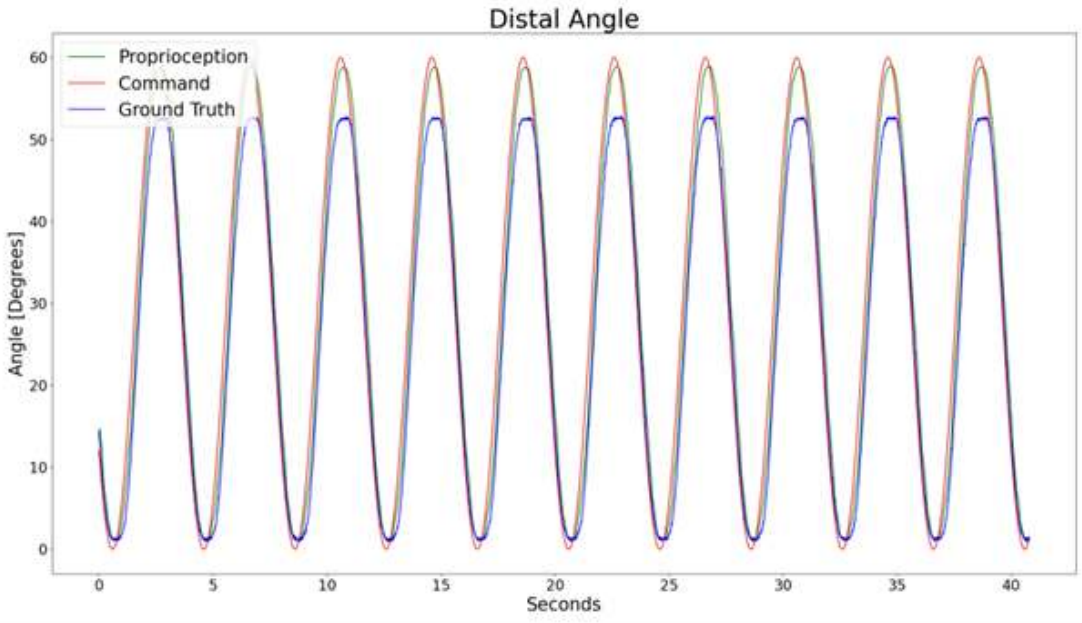
distinction between finger flexion and extension has a limited impact on proprioception accuracy, mirroring the findings related to latency.

The slope of 1.07 indicates that the angles are changing in the same proportion, suggesting that the proprioception measurements closely track the ground truth angles, particularly when disregarding the broader offsets at the signal's lower and upper ranges. Thus, for the purposes of my application, I determined that the current level of proprioception accuracy should suffice. I simply needed to ensure proper calibration and strong tendon tightening. Nonetheless, this analysis highlighted the potential for substantial improvement in proprioception, suggesting that future iterations of the Faive model could benefit from more accurate proprioception methods like angle decoders, which wouldn't be affected by issues like slack.

With this, I had completed the necessary benchmarking for my work and was now prepared to proceed with the next significant phase: integrating the acquired insights into the Faive simulation model and initiating its training process.

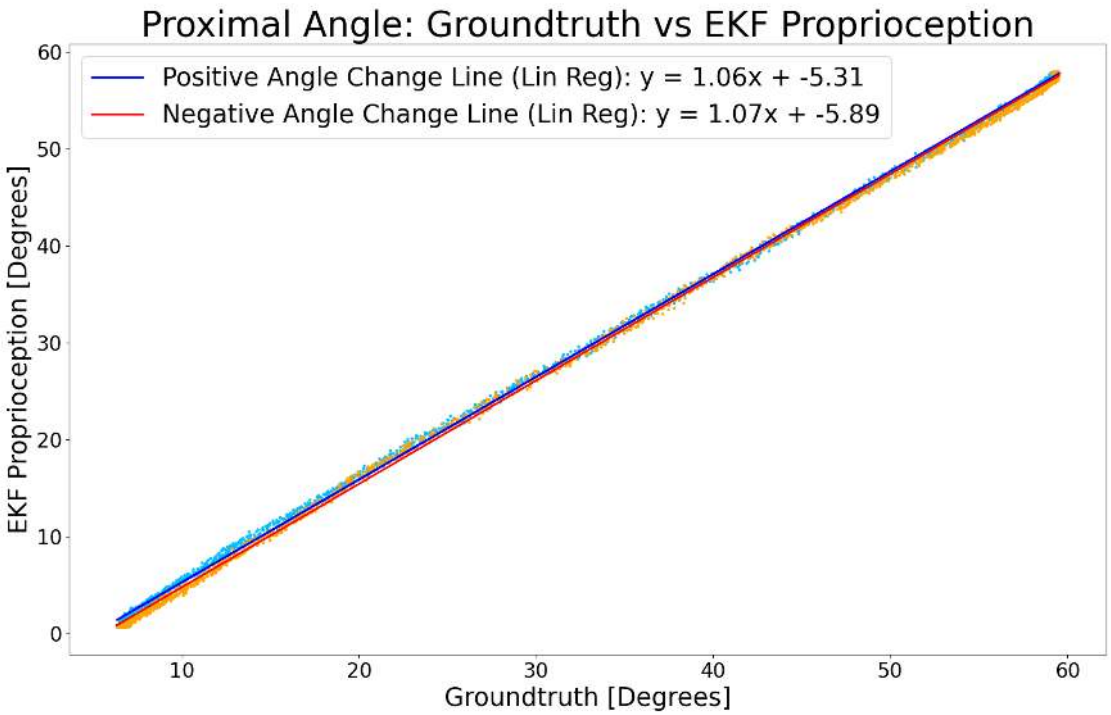


(a)

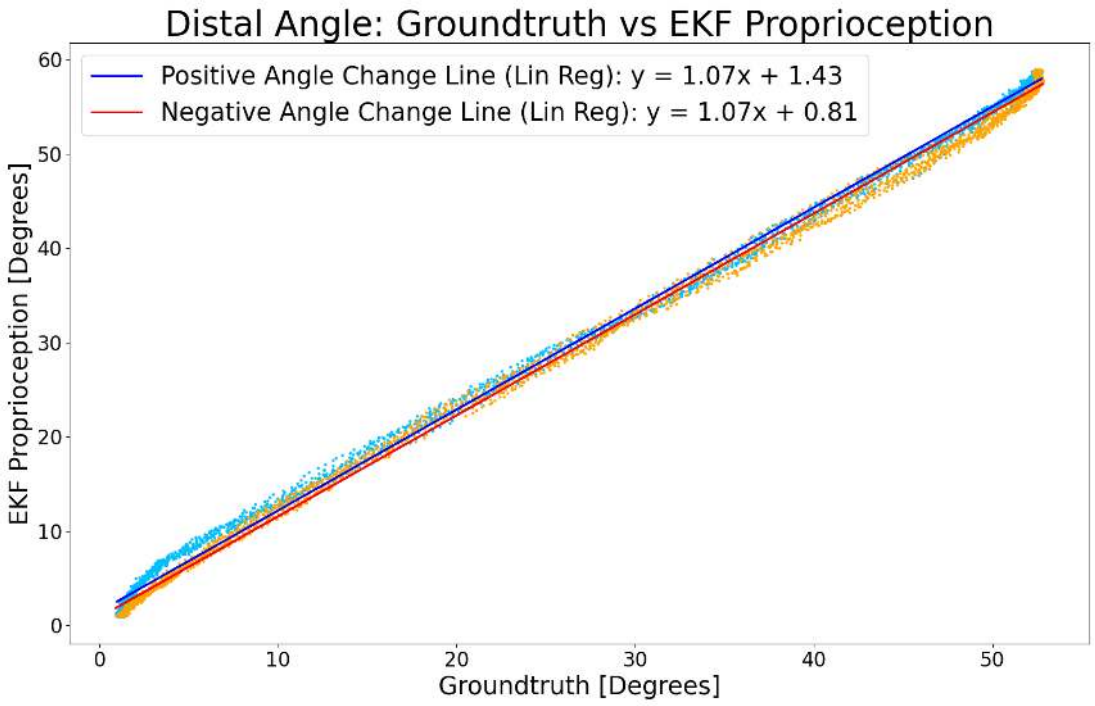


(b)

Figure 37: Sine wave motion of the (a) proximal joint and (b) distal joint, depicted with Commanded, Proprioception, and Ground Truth angles. Notably, the proximal joint never reaches the commanded full extension, while the distal joint remains well below the commanded 60 degrees, highlighting inherent limitations in the Faive framework



(a)



(b)

Figure 38: Comparison of Proprioception Angle against Ground Truth Angle for (a) the proximal joint and (b) the distal joint. A slight hysteresis is evident, indicating a minor dependency on whether the finger is in a flexion (orange data points) or extension (light-blue data points) movement. Notably, the points exhibit minimal scatter, and the slope of the fitted lines approximates one, indicating robust proprioceptive accuracy.

4 Training in Simulation

4.1 IsaacGym Acquaintance & Reward Shaping

After the benchmarking of the physical Faive hand, it was finally time to train it in simulation for dynamic catch and throw behavior. For that, I first needed to get acquainted with the isaacgym simulation implementation of the Faive hand within the lab's framework. Additionally, I had to become familiar with two essential tools for the training in simulation: AWS[21] and WandB[22].

- **AWS**, short for Amazon Web Services, is a popular cloud computing platform offered by Amazon. It provides a wide range of cloud services, from computing power to storage and more, enabling users to run applications and perform various tasks without the need for physical hardware.
- **WandB**, short for Weights and Biases, is a tool developed by OpenAI. It's used for experiment tracking, visualization, and collaboration in machine learning and deep learning projects. WandB helps researchers and data scientists keep track of their experiments, monitor model performance, and share their results with others,

Training numerous hands simultaneously in a simulation on my personal device would have been an incredibly time-consuming task. However, by leveraging AWS, I was able to significantly accelerate the training process. A full simulation run consisting of 5000 training epochs and involving 4096 parallel hands in simulation could be completed in just one hour using AWS.

WandB also played a key role, as it enabled numerical and visual comparisons between various simulation runs. These comparisons involved the examination of numerical data and graphs, providing an overview of rewards and successes over time as well as video excerpts that gave me insight into the tactics that the hand learned during simulation so that I could appropriately reward or penalize those tactics[23] for future simulation runs.

With these two tools in place, I was ready to implement the initial rewards in my reward shaping process. My objective was to strategically reward and penalize specific actions performed by the simulated robotic hands in order to teach them to throw a ball into the air and catch it using only the proximal and distal finger joints' actuation capabilities.

I built upon an existing implementation that already incorporated some rewards for teaching the hand how to reorient a ball inside its grip. One pre-existing penalty (**drop_penalty**) was assigned for letting the ball drop, immediately imposing a substantial negative reward on the Faive hand and resetting the environment, i.e. the finger and ball position. Additionally, I adopted actions (**action_penalty**) and torque (**torque_penalty**) penalties from the existing codebase to make finger movements smoother and less shaky. Obviously, these penalties alone were insufficient for teaching the hands to perform a throwing motion.

Therefore, expanding upon these pre-existing rewards, I implemented the following additional rewards:

- **Reward for Upwards Ball Movement (go_up_reward):** The actor (or Faive hand) received a reward whenever the ball showed a positive velocity in the z-direction (Fig. 39). The greater the speed, the higher was the reward. This encouraged the hand to elevate the ball with a reasonable speed.
- **Reward for Ball Reaching a Defined Height (reach_goal_reward):** The actor was rewarded when the ball reached a specific height above the hand palm (Fig. 40). Successful throws above this height filled a success buffer, which played a significant role in the later stages of training and which reset when the ball dropped to the floor. It was important to define the height above the ball's starting spawn position (self.goal_pos) to prevent the hand from purposely resetting the ball by dropping it on the floor to get rewards. Additionally,

[21] https://aws.amazon.com/
[22] https://wandb.ai/site
[23] The agents' goal is to achieve the highest possible total reward score through their actions. Rewards positively contribute to the score, while penalties have the opposite effect.

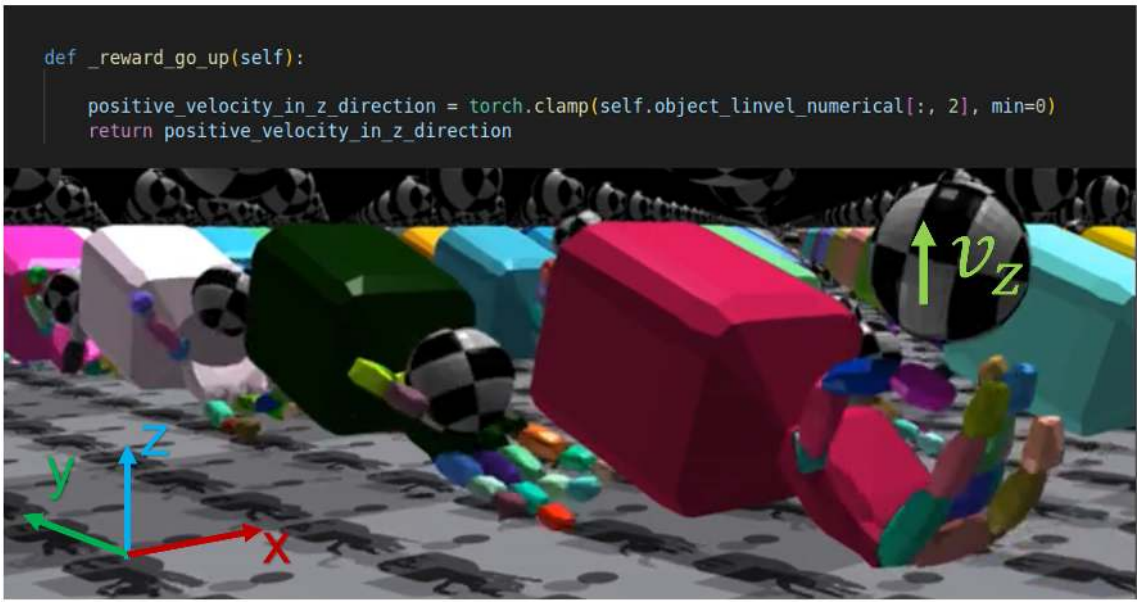


Figure 39: The higher the speed of the ball in positive z-direction, the bigger the reward.

precise tuning of the height value was necessary to ensure that the hand didn't throw the ball too high or too low.

- **Penalty for Overthrowing Behind the Hand Wrist (y_dist_penalty):** To counteract a problem where the simulated hand learned to stabilize ball-catching by exploiting the front side of the box containing the dynamixel servos, I implemented a substantial penalty for allowing the ball to move too far behind the wrist. Unlike the simulated hand model, the real hand has numerous tendons in front of the box (Fig. 26). Therefore, it was essential to penalize this clever strategy during simulation. This penalty operated in a manner similar to the dropping penalty, resetting both the ball's and fingers' position and the success buffer within the environment if the ball was thrown too far into the y-direction (Fig. 41).

The training process included multiple iterations of reward shaping. This reward shaping process involved running numerous training sessions with slightly different parameters in parallel using AWS and comparing the results directly in WandB. Additionally, I implemented bash scripts to automate the training process, allowing for round-the-clock training, even during the night. By isolating each parameter during each shaping iteration, I found the well-working reward weight tuning parameters shown in Table. 4. These parameters were then used in many runs for validation, with the best-performing run's behavior and graphs presented in Fig. 42, 43, and 44.

Reward Name	Value
action_penalty	-0.005
drop_penalty	-4.1
go_up_reward	12.0
reach_goal_reward	50.0
torque_penalty	-0.02
y_dist_penalty	-10.0
acceleration_penalty	0
speed_penalty	0

Table 4: Reward Weighting Parameters: These values have been determined through extensive simulation runs and tuning. Notably, in order to promote fast dynamic behavior, no penalties for high finger acceleration or speed were assigned.

Throughout the reward shaping process, I also needed to become more proficient with parallel

```
    self.success_condition = torch.gt(self.object_pos[:, 2] - self.goal_pos[:, 2] - 0.015, 0)
    # Set self.success_buf based on successful_attempt flag and success_condition
    new_successful_attempt = self.success_condition & ~self.successful_attempt
    self.successful_attempt |= new_successful_attempt
    self.success_buf = new_successful_attempt

    # Reset successful_throw flag when the object is below the startpos value
    reset_condition = torch.lt(self.object_pos[:, 2] - self.goal_pos[:, 2], 0.015)
    self.successful_attempt &= ~reset_condition
def _reward_reach_goal(self):

    return self.success_buf
```

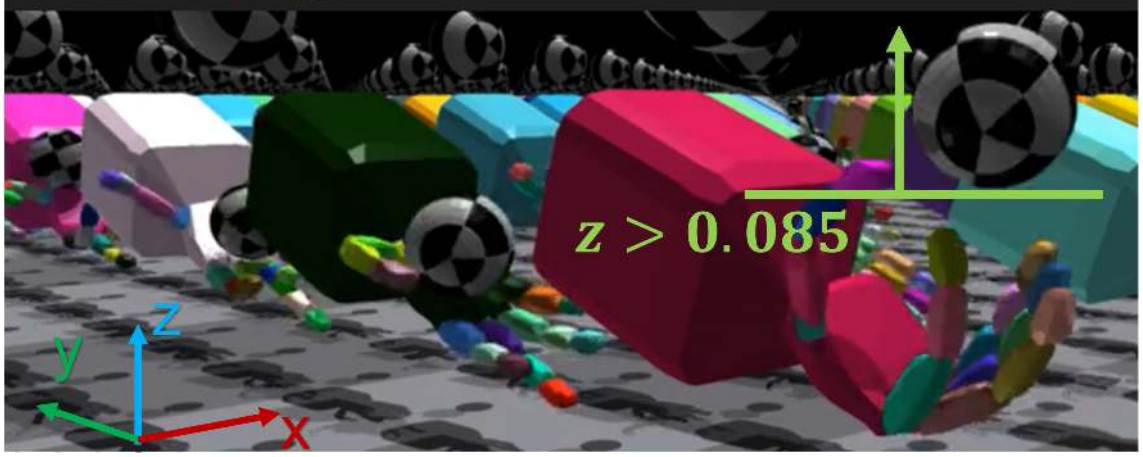


Figure 40: When the ball's center crosses the green line, a big reward is given, and the success_buffer (*self.success_buf*) is incremented for the specific hand. As seen in the code, for the hand to receive this reward again by bringing the ball above the line, the ball must have been below the line or reset before. The starting spawn position of the hand has been tuned to be 0.07m above the ground.

programming in Python. As I wanted to compute states, actions, buffers and rewards of 4096 simulated hands at the same time, conventional for loops would have resulted in very slow epoch training rates (Fig. 45). Therefore, to enhance efficiency, I delved into basic parallel programming techniques to get rid of all for loops that I implemented initially, which significantly accelerated the training rates as depicted in Fig. 46.

4.2 Latency Curriculum Implementation & Open-Loop vs. Closed-Loop

Another important aspect I aimed to incorporate into the simulation model was the consideration of latency in the robot hand's actions. Up until this point, all training had been conducted without any latency between observations and corresponding actions. In simulation, where the hand's actions occurred at a frequency of 20 Hz, I decided to introduce a latency of 0.05 seconds, based on the findings from the latency benchmarking discussed in Chapter 3.5 and depicted again in Fig. 47. This meant that the actions resulting from observations processed by the trained neural network were executed exactly one time step later in the simulation.

However, the immediate introduction of latency didn't yield favorable results in all training runs (Orange Graph in Fig. 48). Therefore, I devised a curriculum that utilized the previously mentioned success buffer *self.success_buf*.[24]

[24]Each time a hand successfully performs a throw above a certain height, a tensor named *self.successes* is incremented for that hand's instance (*self.successes += self.success_buf*). If the environment needs to be reset due to the ball dropping or moving too far in the y-direction, *self.successes* is reset to zero for that instance. Thus, *self.successes* tracks the number of consecutive successes for each hand at any given time.

When the number of consecutive successes for a hand reaches a predefined value, the latency curriculum is activated. Latency is introduced to that specific hand instance, and the hand must adapt to this new challenge. For the examples provided, the number of consecutive successes required to activate the hand's latency adaptation was tuned to six consecutive successful throws and catches.

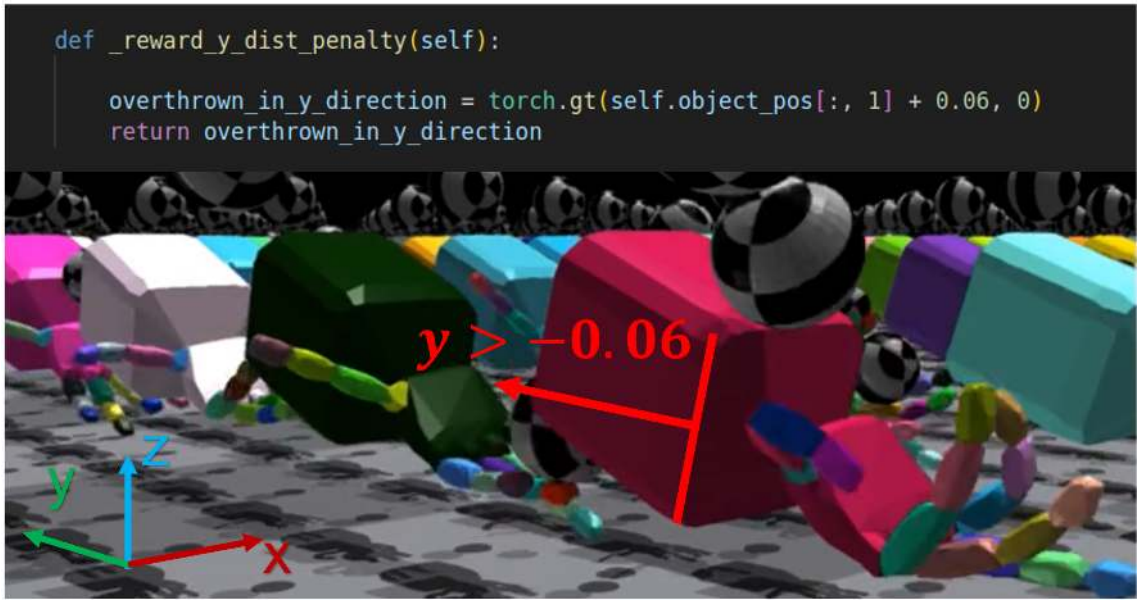


Figure 41: If the ball moves too far backward, the environment is reset, and a penalty is imposed. This penalty has been mainly implemented, to prevent the hand to use the box to stabilize the catching process. The handbox's front is located at y=-0.06m relative to the local coordinate system of each hand.

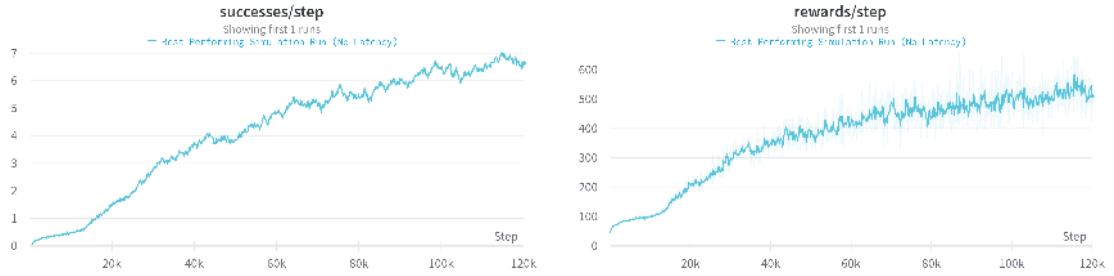


Figure 42: WandB graphs depicting the average number of consecutive successes per step and total rewards per step for the best-performing run. **Left:** After one full training run, the hands achieved an average of 7 consecutive successful throws and catches. **Right:** Thanks to back-propagation based on the PPO algorithm, the average rewards increased steadily throughout the training run.

Initially, each hand started with zero latency at the beginning of the simulation, allowing them to explore the possibility of maximizing their reward by throwing and catching the ball. After a certain number of consecutive successes, those hands began to operate with a 0.05-second latency in their actions (Pink Graph in Fig. 48). This approach provided the actors with an opportunity to first learn the correct behavior before being challenged with latency. Otherwise, the simultaneous introduction of latency and the learning of proper behavior from scratch proved to be overly challenging for the hands in some runs, resulting in poor performance. The curriculum implementation contributed to greater robustness in the training runs and, in the best cases, achieved performances very close to those without any latency, a feat that the method without the curriculum could not come close to.

Moreover, it's essential to note that all these training runs were conducted using an open-loop approach. In this approach, the actor-critic had access to diverse observations, while the actor had only very limited access to these observations. In particular, the actor had no information about the position or velocity of the ball, making the hand essentially blind to the object it was manipulating. Henceforth, I will refer to this approach as the open-loop method. As will be further explored in Chapter 5, closed-loop training runs were also performed, where the actor had access

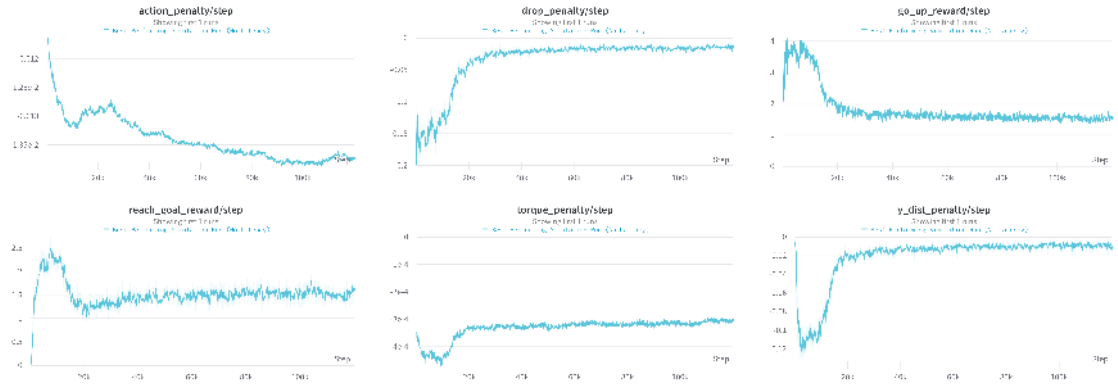


Figure 43: WandB graphs illustrating the progression of rewards based on the hands' behavior for the same run as in Fig. 42. Notably, in the initial stages, steep slopes indicate significant policy changes. After approximately 1000 epochs, most major policy adjustments have taken place, leading to gradual changes in rewards as policy updates become less dramatic per step. The primary objective of the policy is to maximize the total reward, achieved by minimizing penalties (bringing them close to zero) and maximizing rewards (bringing them away from zero) as seen in the graphs. It is important to note that the specific shapes of the graphs vary from run to run, even with the same weighting parameters, because they have varying initial seeds and neural network weights.

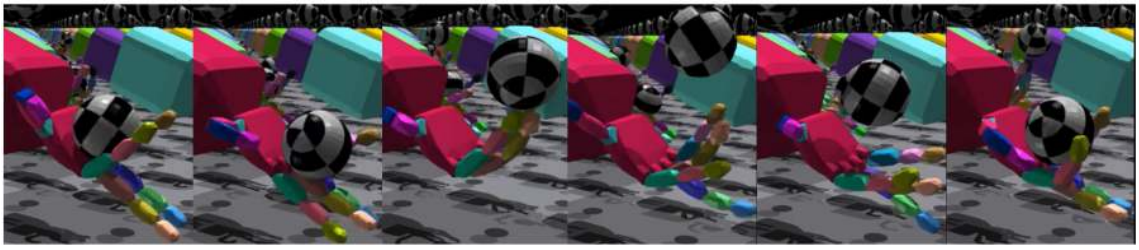


Figure 44: Visualization of the hand's final learned behavior for throwing and catching. It stabilizes the ball during the catching phase using the index and pinky fingers, and actuates all four fingers to elevate the ball into the air.

to object position and angular velocity (Fig. 49). Lastly, it is worth highlighting that every run in this training phase also included the crucial element of Domain Randomization (DR). DR involves the randomization of physical parameters such as object weight, size, friction, finger dynamics, hand palm form, and various other factors that might differ between the simulation and reality. As numerous works discussed in Chapter 1.3 have demonstrated, Domain Randomization plays a pivotal role in facilitating the successful transfer of policies trained in simulation to real-world applications.

```python
def _reward_y_dist_penalty(self):

    #for loop instead of parallel code
    for i in range(self.num_envs):
        if self.object_pos[i, 1] + 0.06 > 0:
            self.overthrown_in_y_buf[i] = 1
        else:
            self.overthrown_in_y_buf[i] = 0

    return self.overthrown_in_y_buf
```

```
fps step: 11061 fps step and policy inference: 10980 fps total: 10127 epoch: 26/5000
fps step: 10818 fps step and policy inference: 10739 fps total: 9410 epoch: 27/5000
fps step: 10949 fps step and policy inference: 10866 fps total: 10020 epoch: 28/5000
fps step: 11168 fps step and policy inference: 11087 fps total: 10220 epoch: 29/5000
fps step: 11220 fps step and policy inference: 11138 fps total: 10260 epoch: 30/5000
fps step: 11134 fps step and policy inference: 11053 fps total: 10182 epoch: 31/5000
fps step: 11049 fps step and policy inference: 10968 fps total: 10087 epoch: 32/5000
```

Figure 45: Using a for loop with 4096 instances for each physics step in simulation is computationally very time-inefficient.

```python
def _reward_y_dist_penalty(self):

    overthrown_in_y_direction = torch.gt(self.object_pos[:, 1] + 0.06, 0)
    return overthrown_in_y_direction
```

```
fps step: 28450 fps step and policy inference: 27930 fps total: 23015 epoch: 26/5000
fps step: 28534 fps step and policy inference: 28007 fps total: 23026 epoch: 27/5000
fps step: 28392 fps step and policy inference: 27866 fps total: 22955 epoch: 28/5000
fps step: 28348 fps step and policy inference: 27829 fps total: 22799 epoch: 29/5000
fps step: 28565 fps step and policy inference: 28041 fps total: 23076 epoch: 30/5000
fps step: 28546 fps step and policy inference: 28019 fps total: 23073 epoch: 31/5000
fps step: 28530 fps step and policy inference: 28005 fps total: 23057 epoch: 32/5000
```

Figure 46: Substituting the for loop with a parallel approach significantly accelerates computation. The terminal output demonstrates that this simple change nearly triples the simulation training speed. Without parallel computation, training time can extend from about 1 hour to roughly 3 hours for a full run. Parallel programming is crucial for environments with numerous similar calculations across multiple instances.

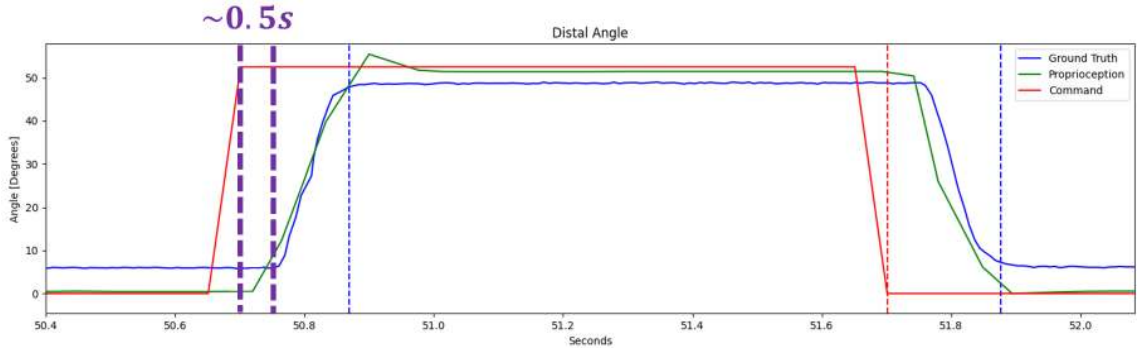


Figure 47: The neural network generates an action for each observation. However, there is approximately a 0.5-second delay before the hand physically initiates this action. Therefore, an effective 0.5-second latency needs to be implemented in the simulation.

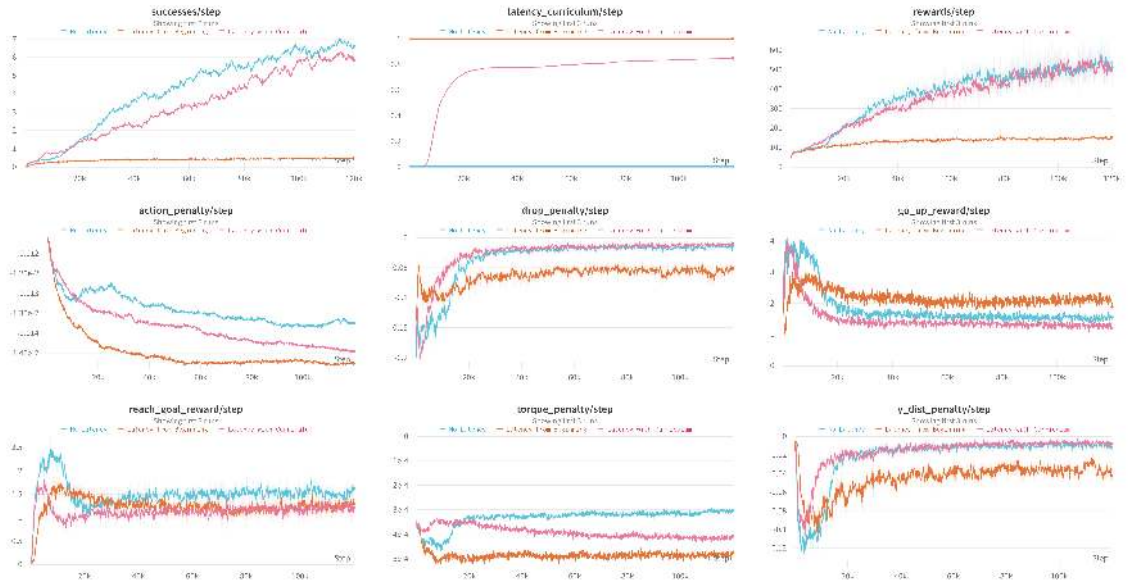


Figure 48: Comparison of Training Runs: We depict training runs without latency (light-blue), with immediate latency introduction (orange), and latency introduced through a curriculum over time (pink). As observed in the upper-left graph, direct latency imposition proved too challenging, resulting in an average of less than one consecutive success. However, introducing latency only after a hand achieved six consecutive successes yielded performance similar to non-latency scenarios. The run with immediate latency introduction generally performed worse across all rewards, primarily due to frequent ball drops, as evident in the central graph. Following the pink line in the top-middle graph , it can be observed how the hands reached the curriculum threshold and transitioned to the latency scenario over time. However, not all of them reached this threshold. It's worth noting that the latency curriculum was designed to be general and thus adaptable for different latency values and for curriculums consisting of multiple levels as well.

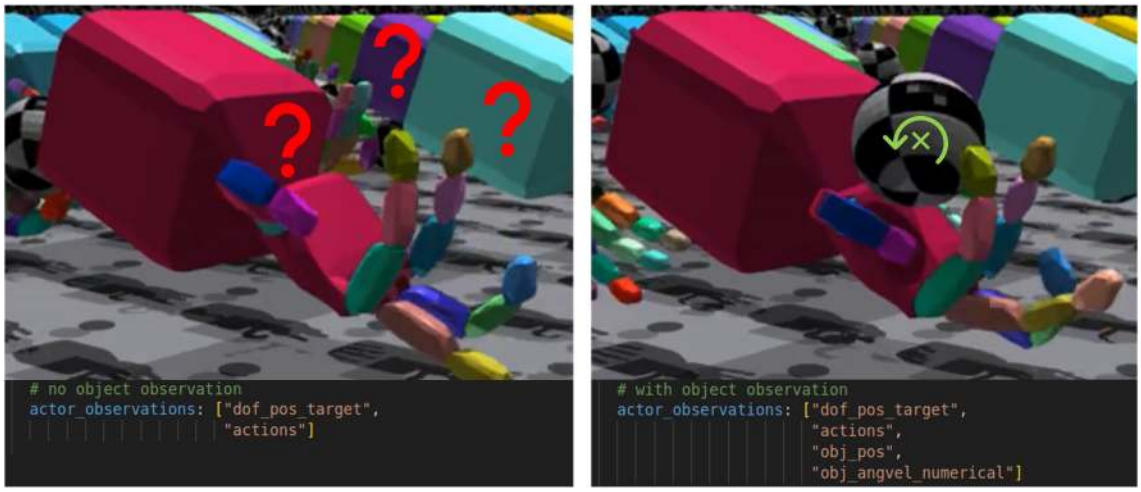


Figure 49: Left (open-loop): In this scenario, the actor lacks any information about the object it is manipulating; it has no cues or inputs regarding the object's state. Instead, it relies solely on information about the target position of each finger and the actions of the fingers. Right (closed-loop): Here, the actor has access to information about the position and angular velocity of the object, which it can use to form its actions. With this information, the actor can more accurately position and move its fingers in response to the object's state at a given time.

5 Initial Stages of Sim-To-Real

5.1 Demonstrating Physical Feasibility

It was fascinating to observe how the robotic hands behaved in simulation while attempting to solve dynamic dexterous tasks. However, simulated environments almost always fall short of replicating reality accurately due to the complexity of real-world objects, their inherent imperfections, and the intricacies of their interactions. As I was interested in the real hand's performance and abilities, it therefore became imperative to explore how the learned behaviors in simulation could be translated to the physical robotic hand. This process is known as the 'sim-to-real' transfer, where behaviors acquired in simulation are executed on the actual robotic setup. Given the substantial gap between simulation and reality, this sim-to-real transfer often presents significant challenges that require special adjustments and methodologies to ensure the successful execution of learned tasks on the physical robot. It is a highly demanding task, which is why my supervisor, Benedek Forrai, and I frequently worked together in this last part of my thesis.

First, we tested whether it was even physically possible to throw and catch a tennis ball with the hand without wrist-flipping. To do this, we utilized the best-performing open-loop behavior learned in simulation (Fig. 42-44) and applied it to the physical hand. Since the hand had no information about the ball's position, it simply repeatedly executed the throwing and catching motion learned in simulation. It took us a lot of attempts until the hand accidentally managed to throw the ball in such a way that it landed back stably in its palm (Fig. 50).

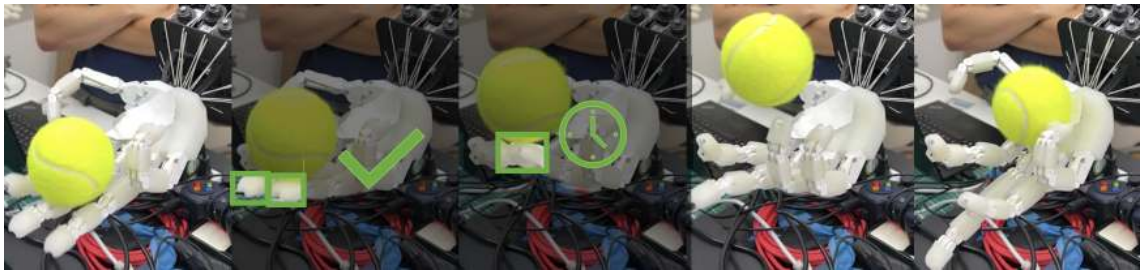


Figure 50: Evidence of the hand's physical capability to execute the catch-and-throw task. Achieving perfect timing between the fingers and the ball is crucial to ensure a precise landing in the hand palm.

Quickly, we realized that the timing of the upward motion of the fingers had to be nearly perfect so that the ball would land in the palm. If it was slightly too slow, the hand would drop the ball (Fig. 51); if it was slightly too fast, the hand wouldn't release the ball in a throwing motion (Fig. 51). Additionally, synchronized motion of the index and middle fingers was crucial to prevent the ball from flying away laterally (Fig. 53).

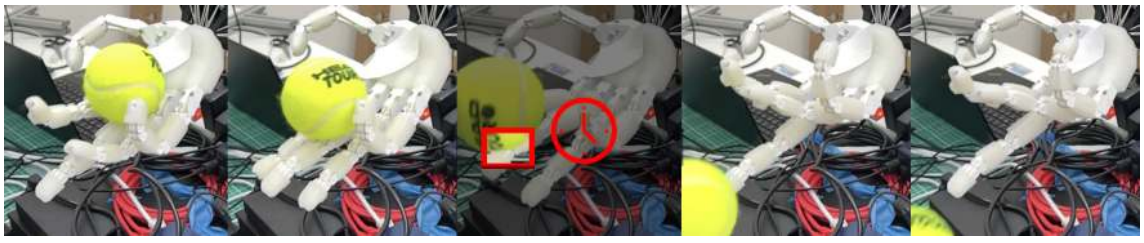


Figure 51: When finger motion is initiated too late, the ball will drop from the hand in distal direction.

Consequently, while we had demonstrated that the motion was technically possible with the real hand, we understood that achieving perfect timing between the ball rolling over the fingers and the initiation of finger movement was essential. This realization made it evident that we needed to equip the real setup with real-time information about the ball's position. Only with this information could we precisely time the upward motion of the fingers to ensure that the ball would ascend and land exactly in the palm of the hand again.

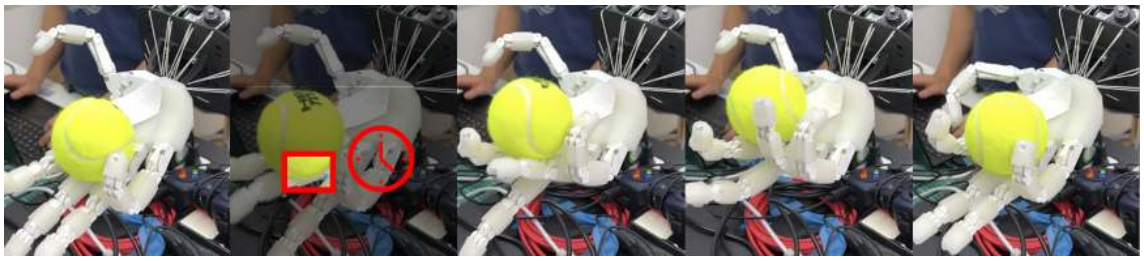


Figure 52: When finger motion is initiated too early, the ball lacks the necessary force to be launched from the hand in a throwing motion.

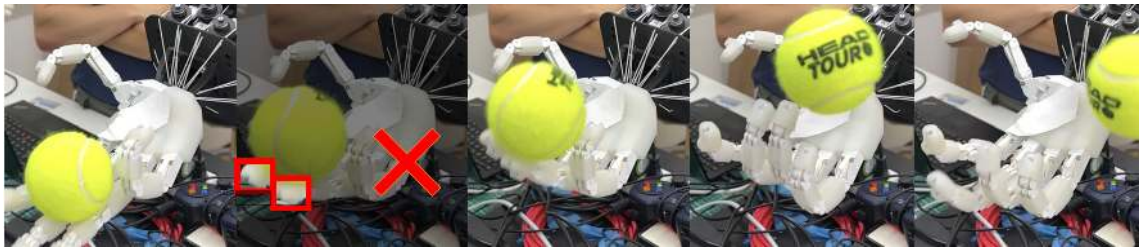


Figure 53: Poor timing between the index and middle fingers results in a lateral force on the ball, preventing it too from landing robustly in the palm.

This led me to contemplate how we could track the real-time position of a sphere with sufficient precision and update rate, even when it was occasionally obscured by the hand and its fingers. It became evident that using apriltags or other markers was not the ideal solution. Since we were dealing with a spherical surface, attaching apriltags would have distorted them significantly. Even if I included flat sections on the sphere's surface, I would have needed a substantial number of tags to robustly determine the ball's center position. Fortunately, I found a more straightforward solution: We could utilize the Qualisys Motion Capture System already available in the lab (Fig. 54).

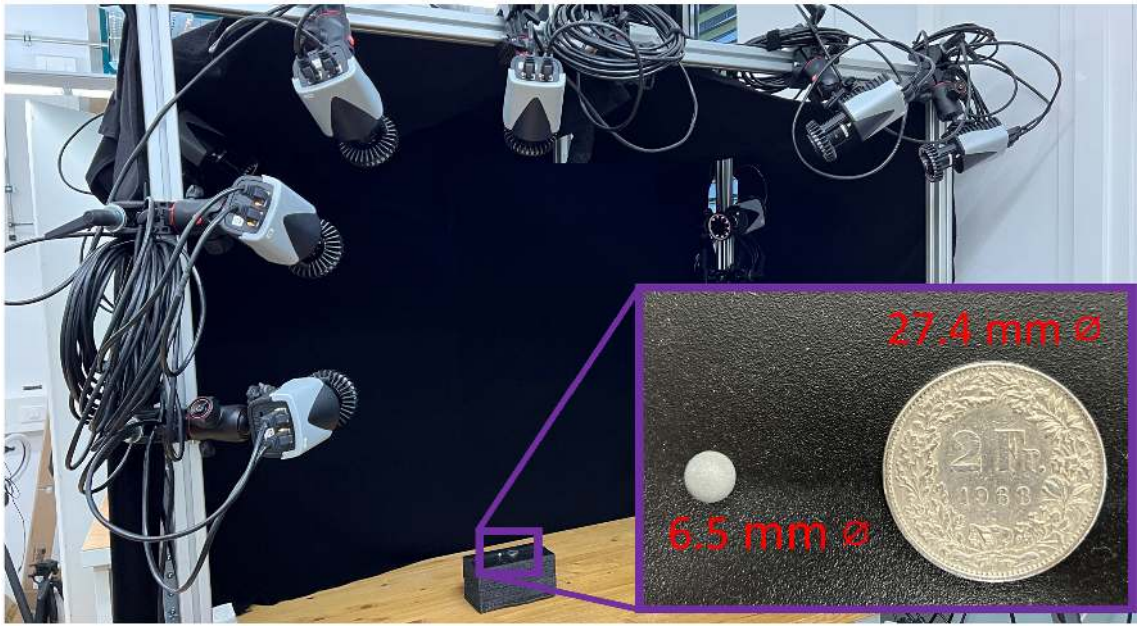


Figure 54: Qualisys Motion Capture System: 8 calibrated cameras are positioned at various angles, observing a 3D space where reflective Qualisys markers are detectable. I utilized the smallest markers available with a diameter of 6.5mm, which can be easily mounted onto screws.

5.2 Object Tracking for Closed-Loop Control

The idea was as follows: By attaching numerous round reflective Qualisys markers to the sphere and tracking their positions using multiple cameras from the MoCap system, we could theoretically track the sphere itself as a rigid body using Qualisys' rigid body tracking software. However, a significant portion of the sphere was consistently obscured by the hand's palm and fingers, making it impossible to determine the ball's position and orientation solely through that software. The observed marker patterns continually varied due to the frequent obscuration of markers, and the software couldn't handle this robustly over extended periods.

However, Benedek had a brilliant idea: instead of using the rigid body tracking software from Qualisys, he proposed developing a special RANSAC algorithm. This algorithm could calculate the center of the ball using all the positions of the observable markers at each instance. Operating under the assumption that the markers were equidistant from the center and given the knowledge of the sphere's radius, we could theoretically pinpoint the exact position of the sphere at any moment, even if multiple markers were occluded. While Benedek focused on implementing this algorithm, my responsibility was to fabricate such a sphere with attached markers that would allow us to determine its position and other properties in real-time.

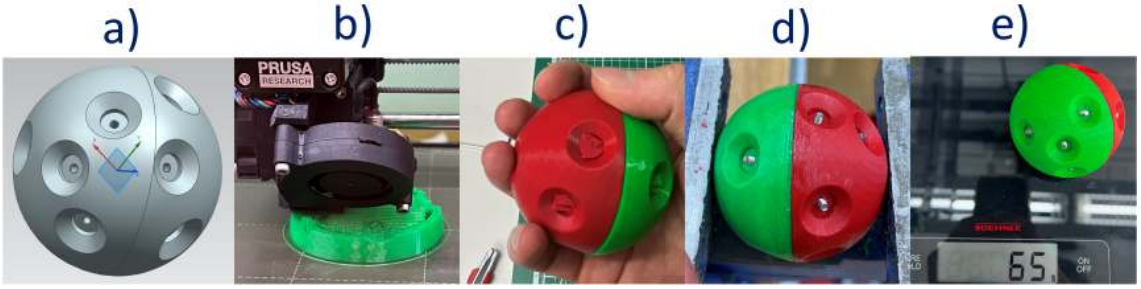


Figure 55: Production Steps of the First Sphere Prototype: a) CAD design with focus on easy printing and assembly. b) 3D printing both sphere halves in parallel to reduce production time. c) Support material removal and screw installation. d) Halves joined with epoxy glue, pressed for 24 hours. e) Validation of sphere dimensions and weight compared to a tennis ball.

Using NX 11, I designed the initial prototype (Fig. 55). I 3D-printed two hollow sphere halves using PETG. These halves featured indentations in which the markers would be placed while maintaining the prototype's spherical shape. The markers could be placed in the indentations by using screws inserted from the inside. After all screws had been inserted, the two halves could be glued together using epoxy glue.

As depicted in Fig. 56, the markers on this first prototype were intentionally distributed asymmet-

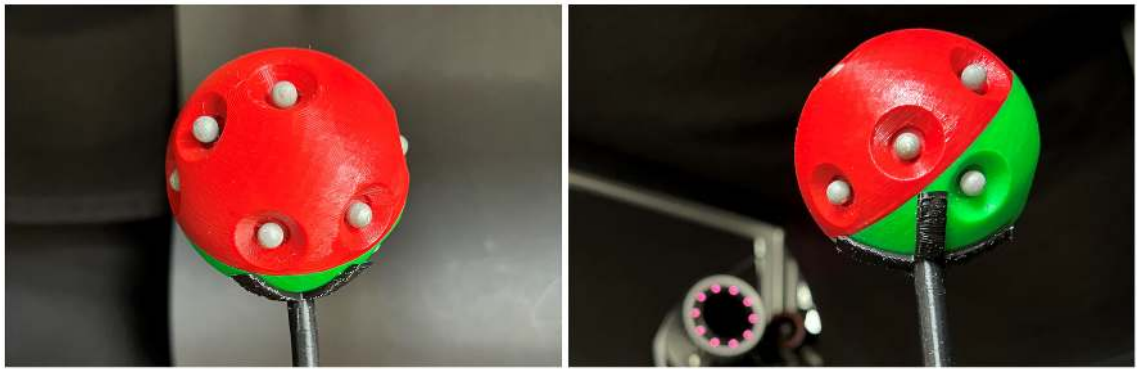


Figure 56: Fully Assembled Sphere with Attached Markers: The markers are positioned so that their outermost points are at an equivalent distance from the center as the sphere's surface. They are distributed asymmetrically and clustered into distinctive patterns.

rically, clustered into distinct groups. Initially, we aimed to measure both the ball's position and orientation simultaneously. However, we soon realized that knowledge about the ball's orientation was unnecessary for our application. We also observed that the position measurement often lacked robustness due to the limited number of markers detectable by the MoCap system, leading to increased measurement noise (Fig. 57).

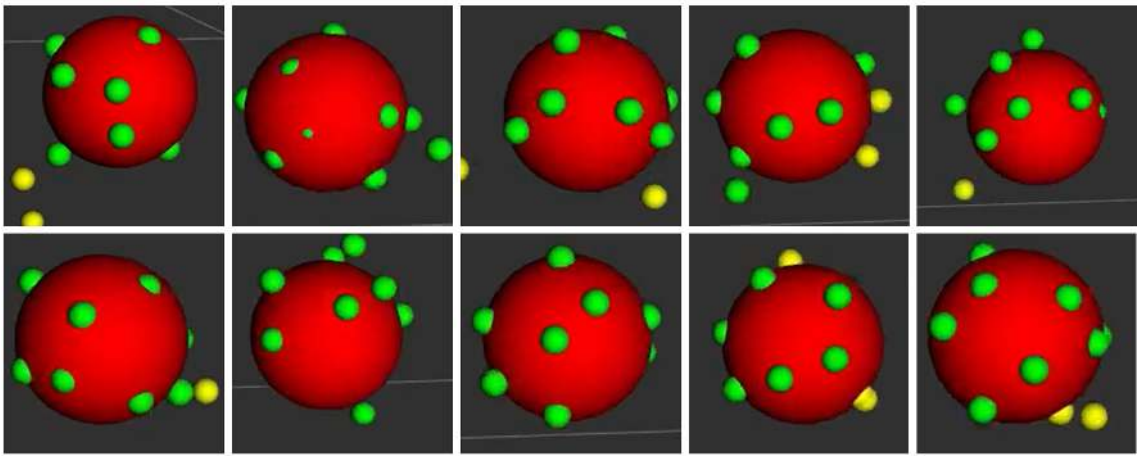


Figure 57: Center Localization of the Red Sphere Using the RANSAC Algorithm in the MoCap Coordinate System. However, due to frequent marker occlusion, position detection of the sphere can become noisy. Outliers (markers in yellow) exert a greater impact on measurements when only a few markers are detected.

Consequently, I designed a new ball with as many markers as possible, evenly distributed across its surface (Fig. 58). This allowed us to achieve highly precise ball position measurements (Fig. 59) and, subsequently, even measurements of the ball's angular velocity. With this setup, we were able to track and transmit the ball's position in ROS at a rate of 100 Hz, matching our requirements for proper closed-loop control.



Figure 58: The new sphere's markers are spaced at a minimum distance of 1 cm to ensure robust detection by the MoCap system. This arrangement allows for a maximum of 28 markers, evenly distributed around the sphere's surface.

But before providing the real hand with the position and angular velocity of the manipulated sphere, we needed to establish the hand's position relative to that sphere. This required defining both the hand's and the ball's positions within the same coordinate frame of the MoCap system. The primary challenge was to determine the handbase position, denoted as Point C in Fig. 60, within the MoCap Coordinate System.

To achieve this, I needed to attach a distinct marker pattern to a location on the hand that could be tracked as a rigid body. The calibration tool, featuring a L-shaped marker pattern, was ideal for this purpose. However, attaching the calibration tool directly to the moving hand was impractical. Moreover, the tool needed to maintain a consistent transform (i.e. orientation and distance) relative to the handbase. To address this, I opted to place the calibration tool onto one of the support beams holding the hand (Fig. 61.left)), identified as Point A in Fig. 60.

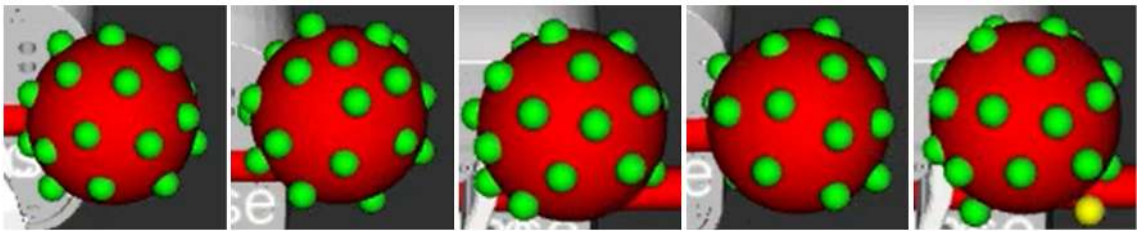


Figure 59: Center Localization of the New Sphere Using the RANSAC Algorithm in the MoCap Coordinate System. The abundance of detected markers at any given instance allows for more accurate center position calculations, minimizing noise even in the presence of outliers (yellow markers).

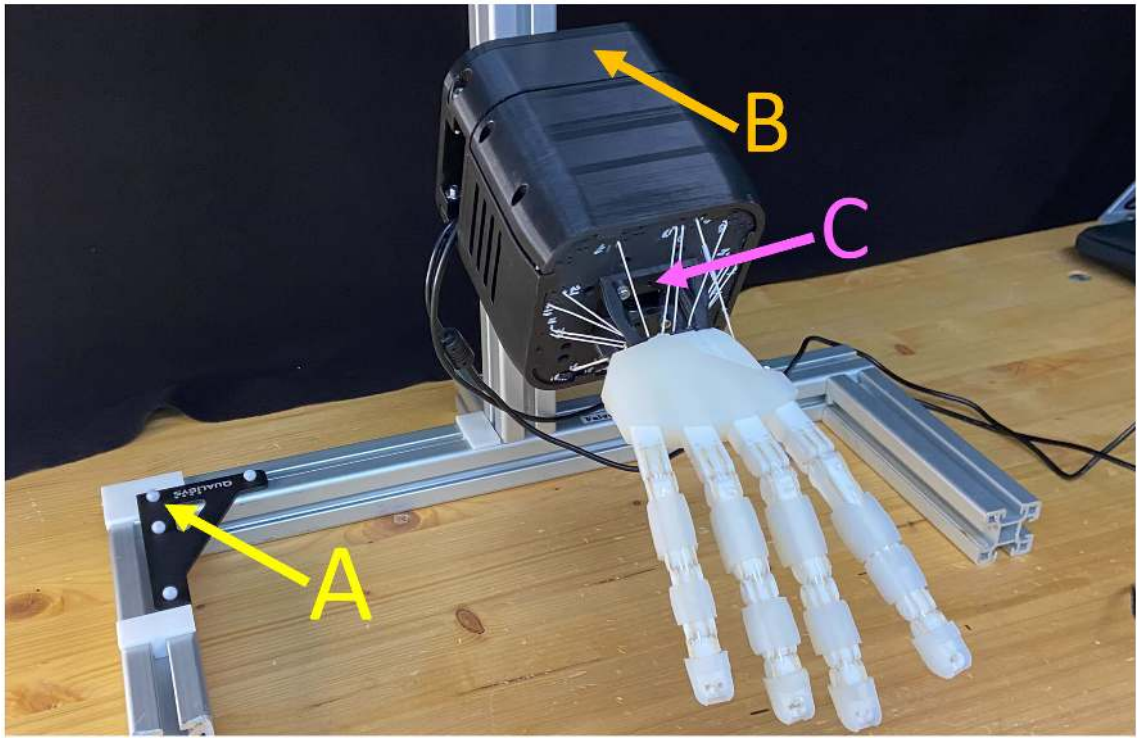


Figure 60: Hand mounted on physical frame. Visualization of the three points A, B, and C that were relevant for the hand transform calculations.

Subsequently, to determine the exact position of the handbase (Point C) within the Mocap coordinate frame, I had to calculate the transform from the calibration tool (Point A) to the handbase. By defining a local coordinate frame at Point A using the orientation of the L-shaped marker pattern of the tool, I could then move the tool and directly measure the relative transform of the tool by using the Qualsys software. However, obtaining the transform from A to C presented a challenge, as the calibration tool could obviously not be placed directly at the handbase.

To address this, after defining the local coordinate system in Point A, I relocated the calibration tool to the top of the handbox (Point B) as depicted in Fig 61.center). This approach enabled me to determine the relative translation in the x, y, and z directions and the relative rotation in roll, pitch, and yaw directions between Points A and B, as illustrated in Fig. 61.right).

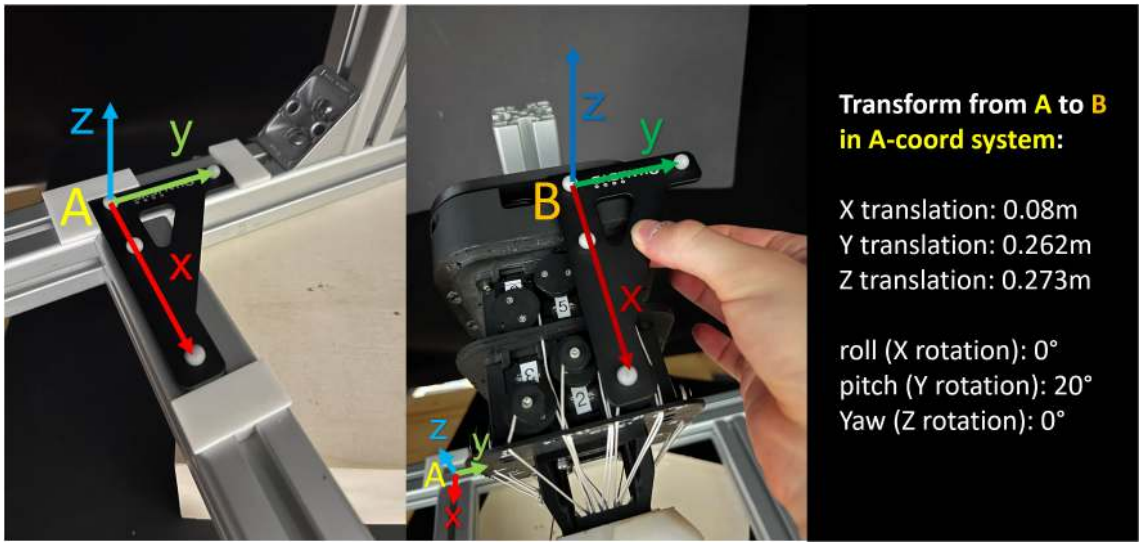


Figure 61: **Left:** The calibration tool is securely fixed using self-designed 3D-printed barriers, ensuring a consistent relative transformation between point A and the handbase (point C). The orientation of the coordinate system is established by the L-form of the calibration tool. **Center:** In the Qualisys program, we can set a local coordinate system based on the tool's position and orientation. We can then move the tool to different locations and obtain the precise translation and rotation of these other points relative to the set local coordinate system. This was done to get the transform of B with respect to the local coordinate system defined at A. **Right:** The top of the handbox is inclined at a 20-degree angle relative to the ground, exactly like in simulation.

$$\text{Translation A} \rightarrow \text{B:} \quad T_{AB} = \begin{bmatrix} 0.08\,\text{m} \\ 0.262\,\text{m} \\ 0.273\,\text{m} \end{bmatrix}$$

$$\text{Rotation A} \rightarrow \text{B:} \quad \begin{cases} \text{Rotation Matrix } R_{AB}\text{: } R_z(0°) \cdot R_y(20°) \cdot R_x(0°) = \begin{bmatrix} 0.9396926 & 0 & 0.3420202 \\ 0 & 1 & 0 \\ -0.3420202 & 0 & 0.9396926 \end{bmatrix} \\ \text{Quaternion } q_{AB}\text{: } \begin{bmatrix} 0 & 0.1736482 & 0 & 0.9848078 \end{bmatrix} \end{cases}$$

However, calculating the transform between Points B and C using this method was still not feasible due to the physical unreachability of the handbase with the tool. But as both points were part of the hand, I resolved this by computing the transform through the CAD model of the hand, as depicted in Fig. 62.

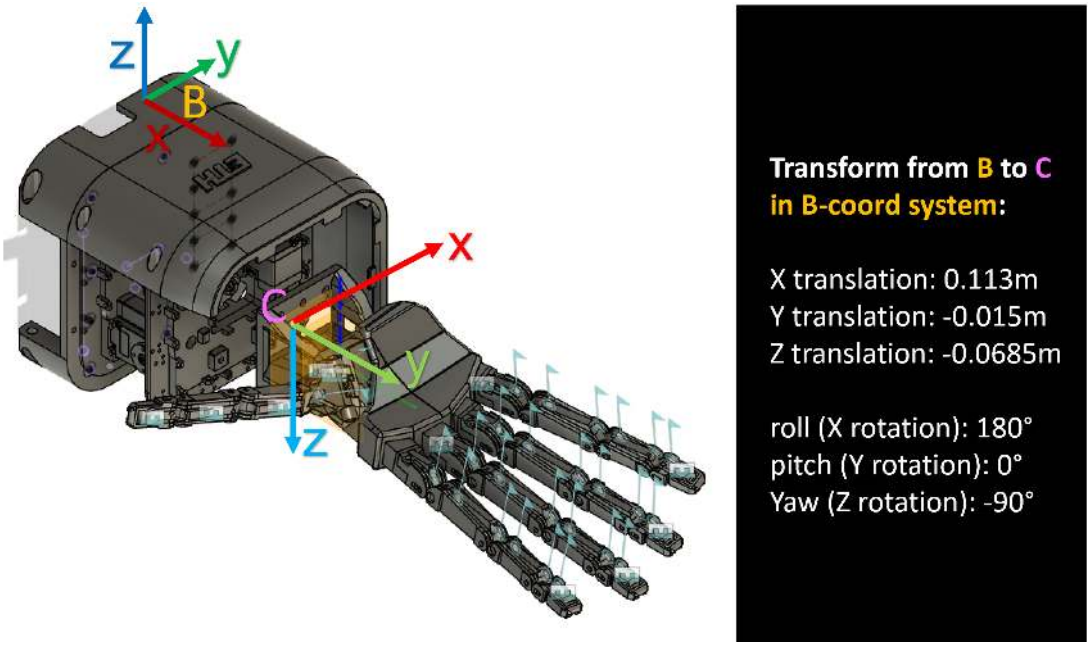


Figure 62: The translation between points B and C in all three dimensions is easily measured in Fusion 360. However, the rotation of the coordinate system in C, predefined in the model, involves multiple axes. To transition from point B's orientation to C, a 180-degree x-axis rotation followed by a -90-degree z-axis rotation is required (using the XYZ-order for Euler Angles). The sequence of these rotations is crucial.

$$\text{Translation A} \rightarrow \text{B:} \quad T_{AB} = \begin{bmatrix} 0.113\,\text{m} \\ -0.015\,\text{m} \\ -0.0685\,\text{m} \end{bmatrix}$$

$$\text{Rotation A} \rightarrow \text{B:} \quad \begin{cases} \text{Rotation Matrix } R_{AB}\text{: } R_z(-90°) \cdot R_y(0°) \cdot R_x(180°) = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \\ \text{Quaternion } q_{AB}\text{: } \begin{bmatrix} 0.7071068 & 0.7071068 & 0 & 0 \end{bmatrix} \end{cases}$$

Thus, in the final step, I combined both transforms, between A and B as well as B and C, to derive the entire transform between Points A and C. This process required gaining an understanding of quaternions[25], rotation matrices, Euler angles as well as rotation orders[26], and how to apply them to obtain the relative transformation between two points, as demonstrated in the following calculations:

[25] Quaternions are a mathematical tool for representing three-dimensional rotations. They offer a more compact and computationally efficient way to describe orientation compared to traditional methods like Euler angles or rotation matrices. Quaternions are widely used in computer graphics, robotics, and physics.

[26] Rotation order refers to the order in which successive rotations are applied to achieve a final orientation. The choice of rotation order can significantly impact the resulting orientation, making it a critical consideration in 3D transformations. Choosing the appropriate rotation order is essential to accurately represent complex orientations and avoid gimbal lock, a limitation associated with some rotation sequences.

$$\text{Translation A} \rightarrow \text{C:} \quad T_{AC} = R_{AB} \cdot T_{BC} + T_{AB}$$
$$= \begin{bmatrix} 0.9396926 & 0 & 0.3420202 \\ 0 & 1 & 0 \\ -0.3420202 & 0 & 0.9396926 \end{bmatrix} \cdot \begin{bmatrix} 0.113\,\text{m} \\ -0.015\,\text{m} \\ -0.0685\,\text{m} \end{bmatrix} + \begin{bmatrix} 0.08\,\text{m} \\ 0.262\,\text{m} \\ 0.273\,\text{m} \end{bmatrix}$$
$$= \begin{bmatrix} 0.16276\,\text{m} \\ 0.247\,\text{m} \\ 0.16998\,\text{m} \end{bmatrix}$$

$$\text{Rotation A} \rightarrow \text{C:} \quad \begin{cases} \text{Rotation Matrix } R_{AC}\text{: } R_{AB} \cdot R_{BC} = \begin{bmatrix} 0 & 0.9396926 & -0.3420202 \\ 1 & 0 & 0 \\ 0 & -0.3420202 & -0.9396926 \end{bmatrix} \\ \text{Quaternion } q_{AC}\text{: } q_{AB} \cdot q_{BC} = \begin{bmatrix} 0.6963642 & 0.6963642 & -0.1227878 & -0.1227878 \end{bmatrix} \end{cases}$$

These solutions were then plugged into our ROS framework and after displaying both the hand and the sphere using their respective transforms within the global MoCap coordinate system, it became evident that both were correctly positioned relative to each other (Fig. 63).

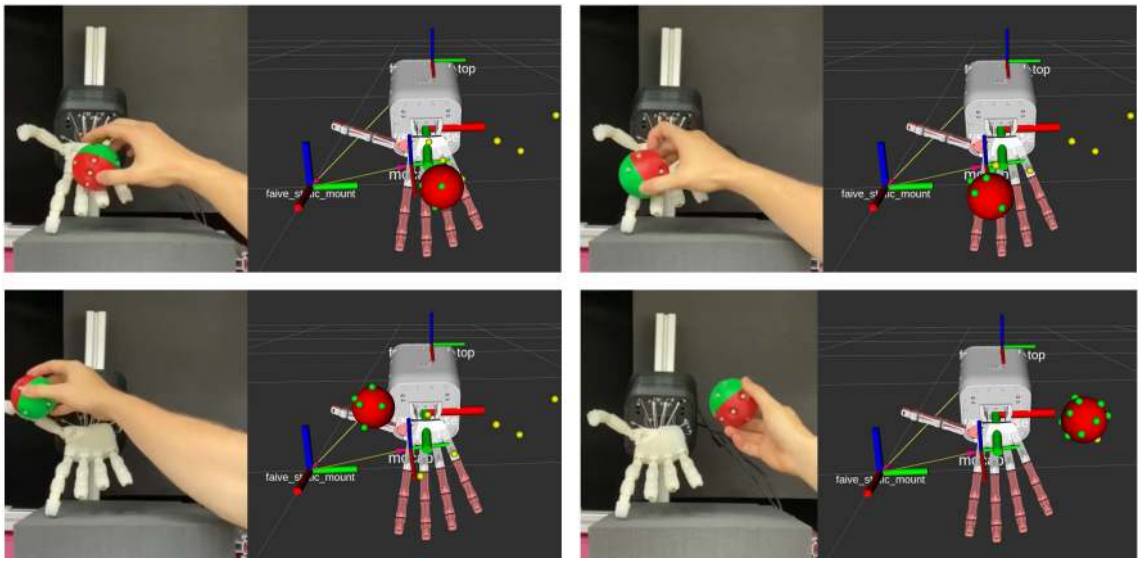


Figure 63: Validation using the previous sphere model to confirm the accurate calculation of the handbase transformation. Both the hand and the sphere are now perfectly aligned, enabling closed-loop actions based on the ball's position and angular velocity.

Unfortunately, it was no longer feasible in the scope of the thesis to test how the real hand would perform in a closed-loop setup. As seen in Fig. 64, closed-loop learning in simulation yielded similar results to open-loop learning. However, exploring how this might change on the real hand was beyond the scope of this thesis and was pursued later for the paper that stemmed from this research.

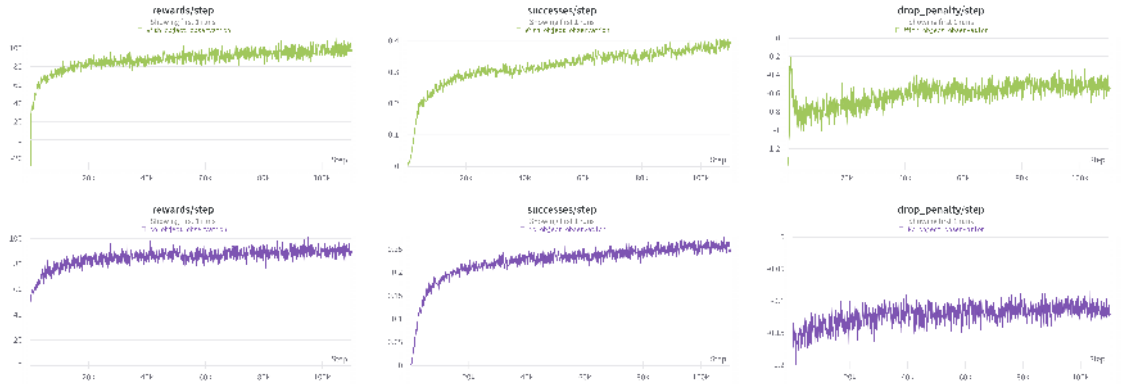


Figure 64: Comparing Closed-Loop (upper) and Open-Loop (lower) Runs: Both runs utilized the same reward weighting parameters and were tuned to be more conservative, featuring high penalties for ball dropping to facilitate the sim-to-real transition. No substantial differences are evident between the two scenarios. However, video observations in WandB indicated that the hand with access to information about the object's state displayed slightly more aggressive actions, resulting in a higher average of consecutive successes (middle), but also a higher average of ball drops (right). This suggests that the greater knowledge about the environment's state provided the actor with a boost in confidence. While the simulation runs were quite similar, it is essential to keep in mind that the differences in reality could be considerably larger.

5.3 Tuning of Tendon Dynamics in Simulation

Nonetheless, there was one last task I wanted to undertake as preparation for the paper before concluding this thesis. During the benchmarking process, I had not only measured the latency and proprioception accuracy of the real Faive hand but I also recorded its dynamic behavior. In simulation, the finger dynamics are simulated by tendons, and the properties of these tendons are defined by two critical parameters: the stiffness[27] coefficient (k_p) and the damping[28] coefficient (k_d). Up to this point, these values had been set somewhat arbitrarily in the simulation. To make the simulated hand more closely resemble the real hand, thus facilitating the sim-to-real process, these values needed to be fine-tuned. This fine-tuning process was initially accomplished through the benchmarking tests I had conducted.

By having the hand perform the same upwards motion of the index finger in simulation (Fig. 65), similar to my benchmarking tests, with varying k_p and k_d values, I could compare the groundtruth proximal angle data obtained from the real hand with the corresponding data obtained from the simulated hands. This allowed me to determine the k_p and k_d values that resulted in the closest match between the dynamic step response in simulation and reality (Fig. 66).

[27] A positive value generates a spring force (linear in position) acting along the tendon.
[28] A positive value generates a damping force (linear in velocity) acting along the tendon.

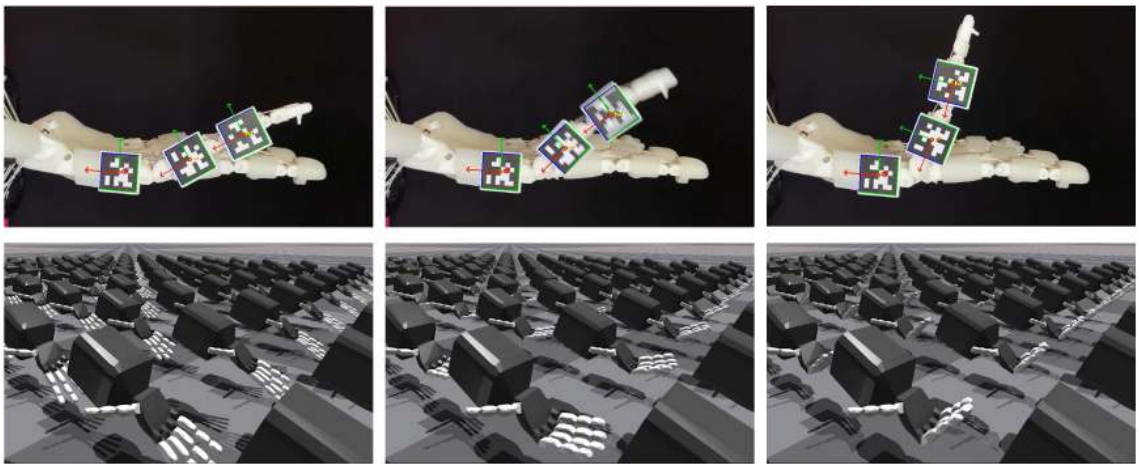


Figure 65: Simulation of 400 hands with varying k_p and k_d coefficients replicating the benchmarked index finger upward motion.

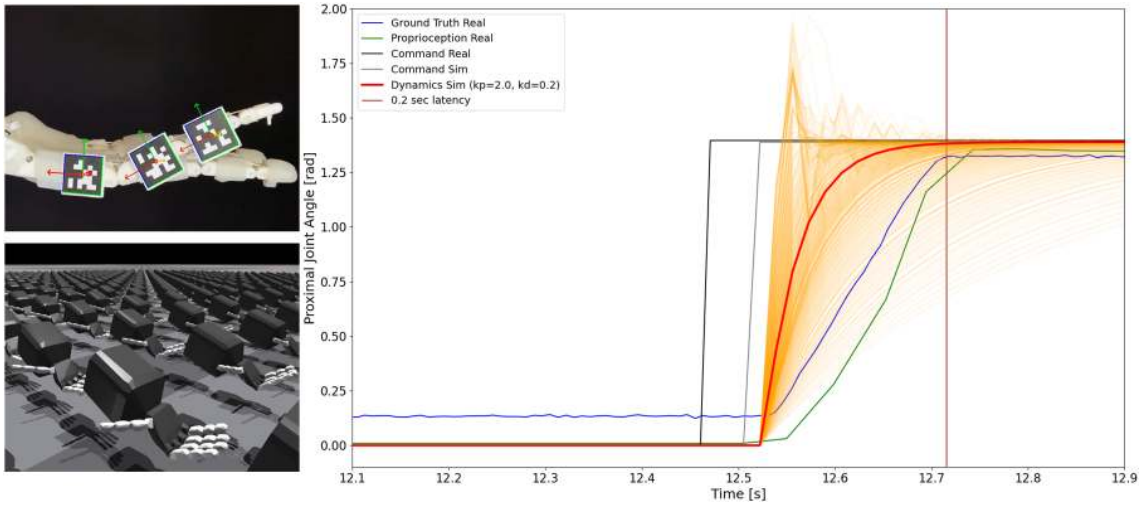


Figure 66: Comparing the dynamic behavior of the real hand (blue) to that of 400 simulated hands with varying k_p and k_d coefficients (orange) and 0.05-second latency implementation. The red graph with $k_p = 2.0$ & $k_d = 0.2$ represents the behavior among the 400 simulated hands that most closely approximates the real-world performance. Notably, the simulated fingers exhibit in general faster initial acceleration, potentially attributable to the absence of slack in simulation.

However, during active finger movements, factors beyond just the stiffness and damping parameters, such as latency and slack, could distort the dynamic behavior between the real and simulated hands. Therefore, it made sense to conduct a second dynamic behavior test to fine-tune the stiffness and damping coefficients in the simulation. This test did not involve active movements initiated by actuation. Instead, we let a tennis ball drop onto the rigid index finger tip when the finger was at a 40-degree incline. This allowed us to measure the hand's passive dynamic behavior in a scenario where slack played a smaller role and compare it to the simulation's response.

For this test, it sufficed to track the relative vertical displacement of the index finger tip instead of the angle change. To achieve this, I attached a Qualisys marker to the finger tip using double-sided tape (Fig. 67). Then, I let the ball drop from a 20 cm height onto the finger tip while recording the marker's position with the MoCap system. As the exact release point for the ball varied in each trial, I conducted multiple consecutive tests per recording to obtain an averaged measurement.

During this process however, I noticed that the hand exhibited some instability as it was not rigidly connected to the handbox, despite being secured with multiple screws. To adress this issue,

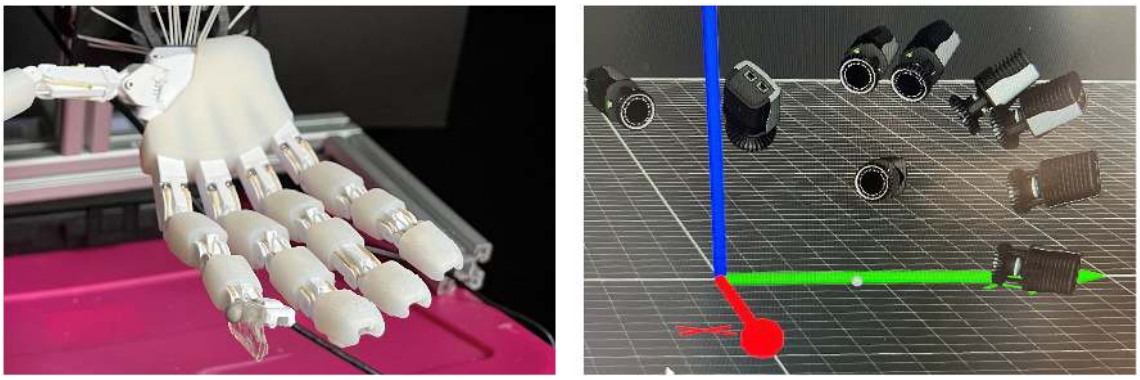


Figure 67: Left: Qualisys marker attached to the finger tip with double-sided tape. Right: 3D position tracking of the marker (white dot) by the MoCap system.

I applied glue to firmly attach the hand to the base, resulting in reduced instability and more reliable measurements of the hand's dynamic behavior, as it eliminated most of the undesirable vertical hand movement (Fig. 71).

In simulation, we replicated the same ball-dropping procedure and measured the finger's vertical motion for a wide range of different k_p and k_d coefficients, (Fig. 68). The graph in Fig. 69 shows the direct comparison of the finger's responses in simulation and reality. Remarkably, the exact same values that we obtained in the first experiment for active dynamics also closely matched the behavior in this second experiment for passive dynamics. Consequently, we had identified well-tuned k_p and k_d values, which we could now use for new reinforcement learning training runs.

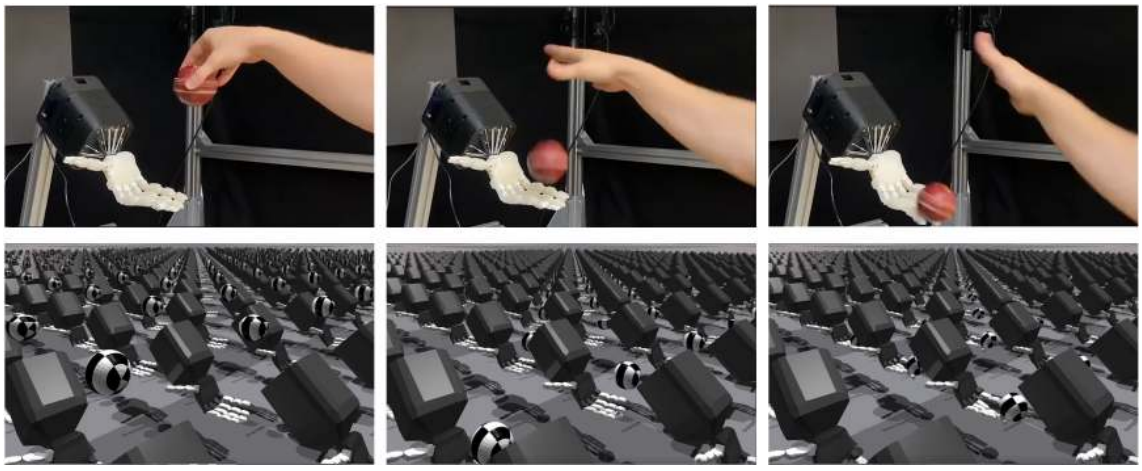


Figure 68: Simulation of 400 hands with varying k_p and k_d coefficients replicating the ball-dropping tests conducted in the real world

When we deployed the newly learned behaviors with the tuned k_p and k_d values on the real Faive hand, we did not observe any significant differences compared to the previous version. However, the updated parameters appeared to enhance overall stability and robustness. With that, the initial stages of the sim-to-real process were concluded. Later, following the completion of this thesis, we recognized that the task of throwing and catching a ball without wrist or hand position movement is extremely challenging to achieve through simulation learning alone due to the need for nearly perfect timing. Factors like slack, latency, and proprioception inaccuracies, as observed in my benchmarking tests, introduce random time delays that are practically impossible to account for in simulation. Consequently, we shifted to other sim-to-real methodologies, such as DreamerV3, an algorithm proposed by Hafner et al. (2023), for the subsequent paper. This transition to other methods was made possible only thanks to the invaluable insights gained in this chapter: the demonstration that the hand is physically capable of performing this complex task, the imple-

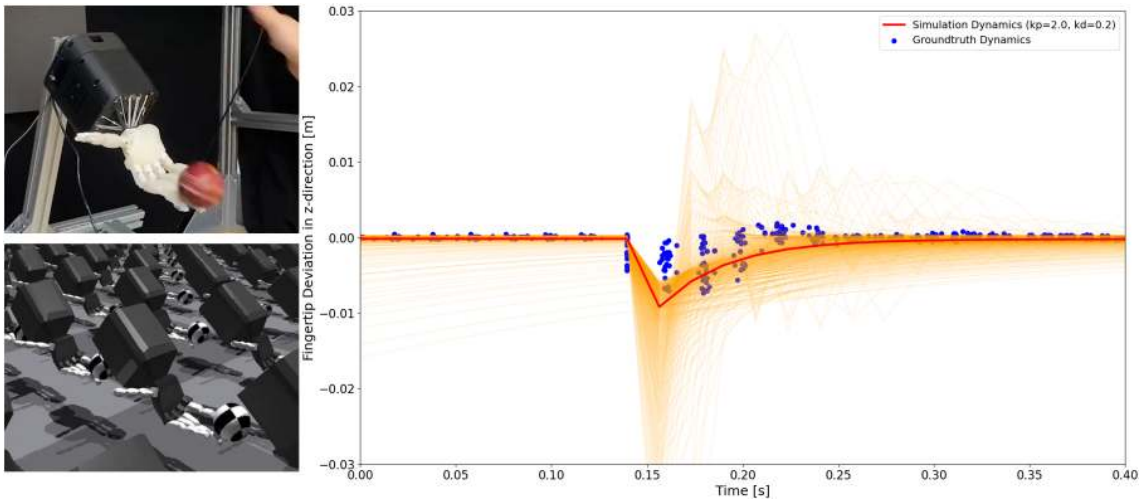


Figure 69: Comparing the recorded dynamic behaviors of the real hand (blue) to that of 400 simulated hands with varying k_p and k_d coefficients (orange). Remarkably, the red graph, representing the best approximation among the simulated hands to real-world performance, employs the same k_p and k_d values as in the first experiment.

mentation of sphere position and angular velocity tracking, and the process of tuning the stiffness and damping values of the simulated hand. So, while I did not finish the sim-to-real process of the ball throw-and-catch task in the scope of this thesis, I established a foundation for future research related to complex tasks and their sim-to-real transfer on the Faive hand.

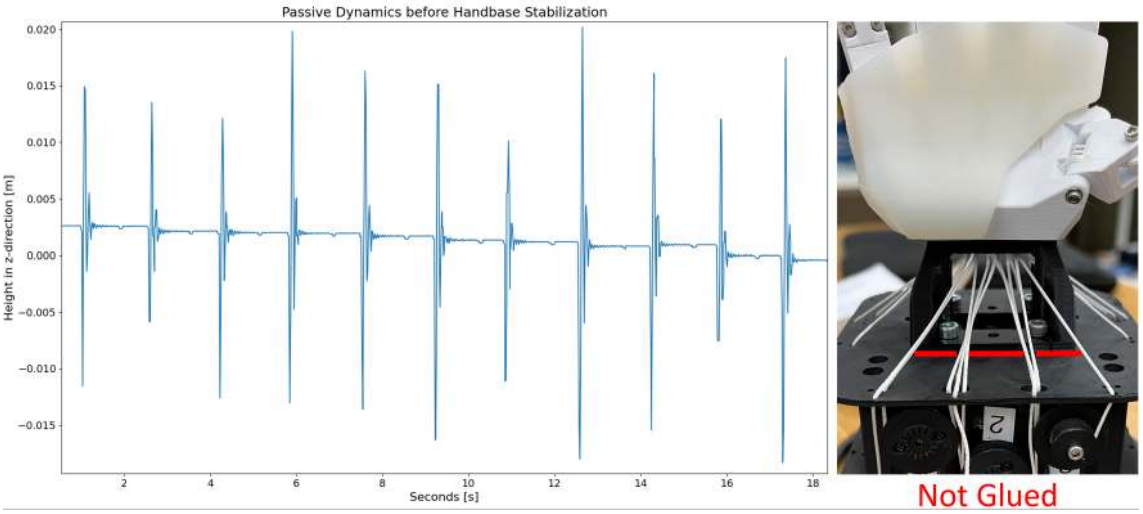


Figure 70: Before Handbase Stabilization: Initial inaccurate measurements of the fingers' passive dynamic behavior were attributed to the lack of rigid attachment between the hand and the handbox. Following the impact of the tennis ball, the entire hand exhibited an excessive relative down-and-up motion (Within a range of ±2 cm), surpassing the smaller more intricate finger movements.

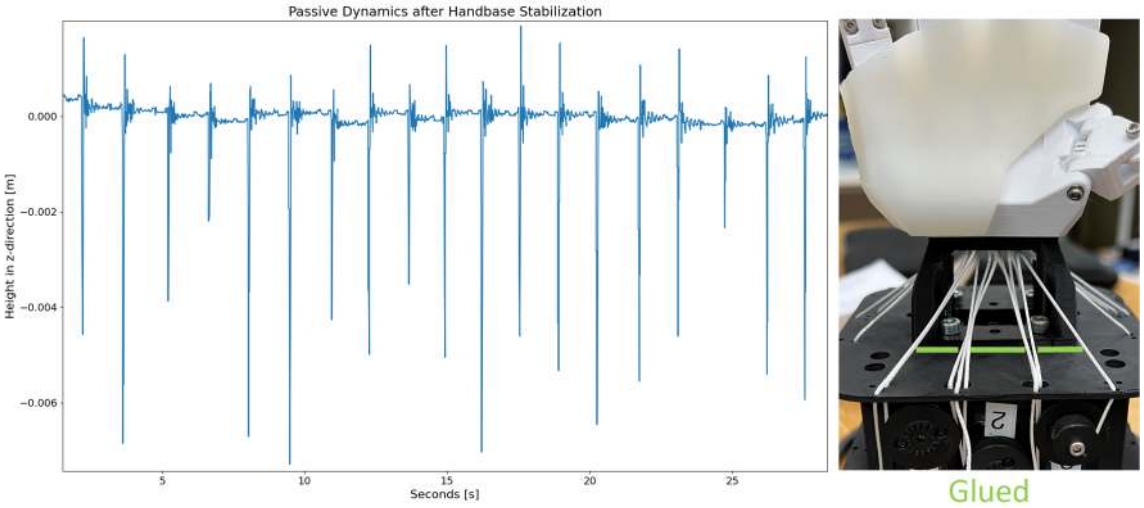


Figure 71: After Handbase Stabilization: Most of the hand's shakiness was eliminated by gluing the hand to the box, allowing for more precise measurement of the fingers' vertical motion (−0.8 cm to 0.2 cm) in response to the falling tennis ball.

6 Conclusion

To summarize, the primary objectives of this thesis were successfully achieved. An easy-to-use and extensible framework dedicated to benchmarking the latency, proprioception accuracy, and dynamics of the real Faive hand was designed. This system identification was long overdue and carried implications not only for the further hardware development of the Faive hand but also for refinements in the Faive hand simulation model. For instance, the extensive benchmarking efforts undertaken in this thesis enabled the reduction of the hand's latency to levels comparable to human reaction times, achieved by optimizing the matrix inversion method within the Faive codebase.

Furthermore, the Faive IsaacGym implementation was enhanced by applying reinforcement learning techniques and reward shaping to train a policy capable of throwing and catching a ball, both with and without knowledge of the ball's state. The incorporation of latency data into curricula allowed for a more realistic replication of the real-world within the simulation. Additionally, insights gained from benchmarking were used to fine-tune the dynamics of the Faive hand's fingers in simulation. These accomplishments brought the simulated hand closer to reality, providing a solid foundation for future policies and manipulation tasks executed by the real hand.

Another noteworthy contribution was the development of a real-time ball position tracking system, an innovation that in this form, to the best of my knowledge, has not been previously documented in the literature. This system alone has the potential to enable a range of new closed-loop skills for the Faive hand.

While dynamic dexterous manipulation on the physical hand was not within the scope of this thesis, it's essential to acknowledge that the work undertaken here represents significant preparatory steps toward this ultimate goal. Presently, ongoing work involves the implementation of DreamerV3 to directly train the real hand for the throw-and-catch task, facilitated by the real-time ball tracking system established within this thesis.

The journey throughout this thesis has been a remarkable learning experience, encompassing both practical and theoretical aspects. I encountered various challenges in hardware and software domains, which demanded innovative solutions. From the design of trackable balls to the creation of specialized apriltag holders, I faced numerous hardware challenges. Additionally, I significantly advanced my proficiency in various software tasks, coding thousands of lines in Python, encompassing data processing filters, algorithms, ROS packages, IsaacGym implementations, OAK D-S2, Qualisys-related scripts, and many other implementations. Furthermore, my exploration extended to the realms of reinforcement learning, computer vision, object tracking, parallel programming,

3D transformations, data synchronization, and benchmarking. It also entailed the acquisition of proficiency in operating systems like Ubuntu, as well as in tools like PyTorch and ROS – areas that were entirely unfamiliar to me before. In parallel with these practical skills, I also conducted theoretical research that expanded my understanding of dexterous manipulation learning for robots and introduced me to various learning methods.

I hope that this thesis serves as an inspiring resource and guide for other students embarking on the path of RL learning for dexterous manipulation, particularly in the context of the Faive hand. In conclusion, I consider this thesis a success, despite the absence of a final, fully demonstrable dynamic dexterous manipulation on the real Faive hand. My overarching objectives for the thesis were twofold: to expand my knowledge of robotic learning and to contribute to the advancement of the Faive hand's development. In my opinion, I have successfully achieved both of these goals. The accomplishments in system integration, latency reduction, RL expansion, simulation enhancement, and ball tracking not only propel future work on the challenging task of teaching the Faive hand to throw and catch a ball but hopefully also hold the potential to benefit all individuals involved in the further development of the Faive hand.

7 Acknowledgements

I wish to express my deep gratitude to the exceptional individuals who played instrumental roles in making this thesis a reality. Without their support, this research would not have been possible in this form.

First and foremost, I owe a debt of gratitude to Benedek Forrai for his outstanding supervision. As the first student under Benedek's guidance, I never once felt lost or unsure. On the contrary, Benedek proved to be an exemplary supervisor, offering insightful guidance, unwavering support, and a wealth of technical knowledge based on his own experiences. His mentorship has been pivotal to the success of this work.

I must also extend my appreciation to Robert Katzschmann, who made it possible for me to embark on this Bachelor thesis journey within the Soft Robotics Lab. It was through his recommendation that I was connected with Benedek, and for this, I am truly grateful. Additionally, I want to acknowledge Yasunori Toshimitsu for his invaluable technical insights regarding the Faive hand.

Last but not least, I wish to express my thanks to my family and friends. Their encouragement, motivation, and genuine interest in this research project have been a constant source of inspiration for me and the driving force behind the successful completion of this thesis.

References

Arunachalam, S. P., Silwal, S., Evans, B., & Pinto, L. (2022). Dexterous imitation made easy: A learning-based framework for efficient dexterous manipulation. (Cit. on p. 3).

Chen, T., Xu, J., & Agrawal, P. (2021). A system for general in-hand object re-orientation. *CoRR*, *abs/2111.03043*. https://arxiv.org/abs/2111.03043 (cit. on p. 6)

Hafner, D., Pasukonis, J., Ba, J., & Lillicrap, T. (2023). Mastering diverse domains through world models. (Cit. on p. 47).

Handa, A., Allshire, A., Makoviychuk, V., Petrenko, A., Singh, R., Liu, J., Makoviichuk, D., Van Wyk, K., Zhurkevich, A., Sundaralingam, B., Narang, Y., Lafleche, J.-F., Fox, D., & State, G. (2022). Dextreme: Transfer of agile in-hand manipulation from simulation to reality. *arXiv* (cit. on pp. 1, 2).

Huang, W., Mordatch, I., Abbeel, P., & Pathak, D. (2021). Generalization in dexterous manipulation via geometry-aware multi-task learning. *CoRR*, *abs/2111.03062*. https://arxiv.org/abs/2111.03062 (cit. on p. 6)

Qin, Y., Huang, B., Yin, Z.-H., Su, H., & Wang, X. (2023). Dexpoint: Generalizable point cloud reinforcement learning for sim-to-real dexterous manipulation. In K. Liu, D. Kulic, & J. Ichnowski (Eds.), *Proceedings of the 6th conference on robot learning* (pp. 594–605). PMLR. https://proceedings.mlr.press/v205/qin23a.html. (Cit. on p. 2)

Qin, Y., Wu, Y.-H., Liu, S., Jiang, H., Yang, R., Fu, Y., & Wang, X. (2021). Dexmv: Imitation learning for dexterous manipulation from human videos (cit. on p. 5).

Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., & Levine, S. (2018). Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. (Cit. on p. 5).

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., & Abbeel, P. (2017). Trust region policy optimization. (Cit. on p. 8).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. (Cit. on p. 8).

Senoo, T., Yamakawa, Y., Mizusawa, S., Namiki, A., Ishikawa, M., & Shimojo, M. (2009). Skillful manipulation based on high-speed sensory-motor fusion. *2009 IEEE International Conference on Robotics and Automation*, 1611–1612. https://doi.org/10.1109/ROBOT.2009.5152852 (cit. on p. 7)

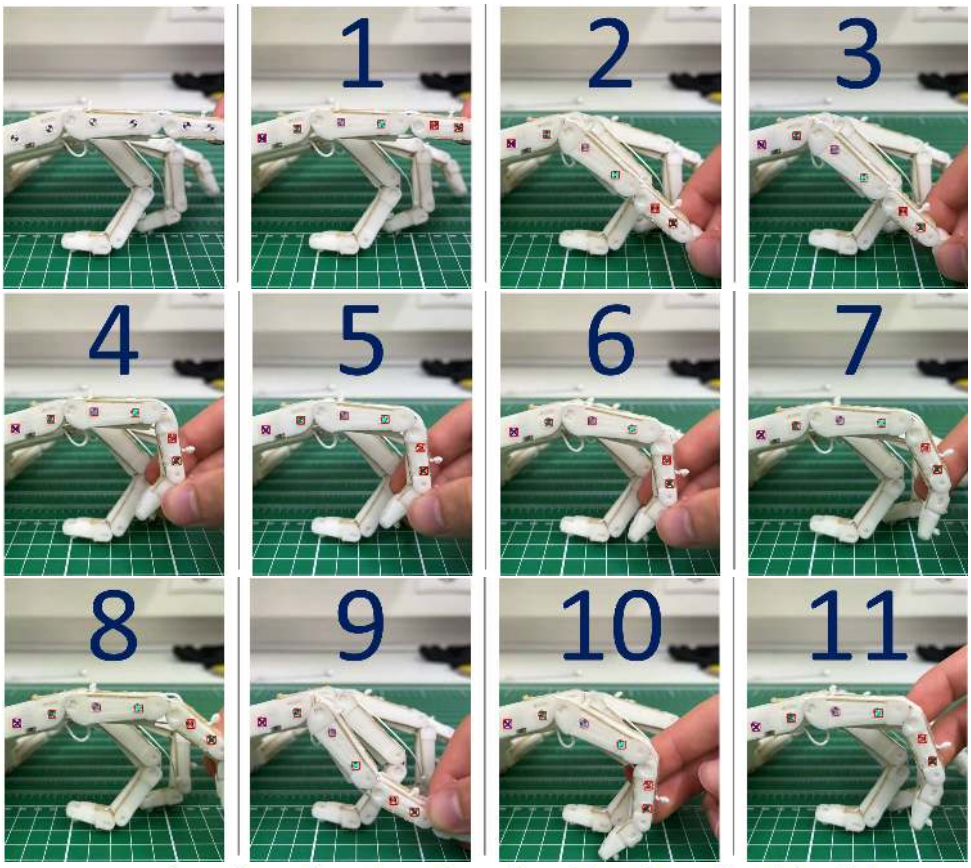
Shaw, K., Bahl, S., & Pathak, D. (2023). Videodex: Learning dexterity from internet videos. In K. Liu, D. Kulic, & J. Ichnowski (Eds.), *Proceedings of the 6th conference on robot learning* (pp. 654–665). PMLR. https://proceedings.mlr.press/v205/shaw23a.html. (Cit. on p. 3)

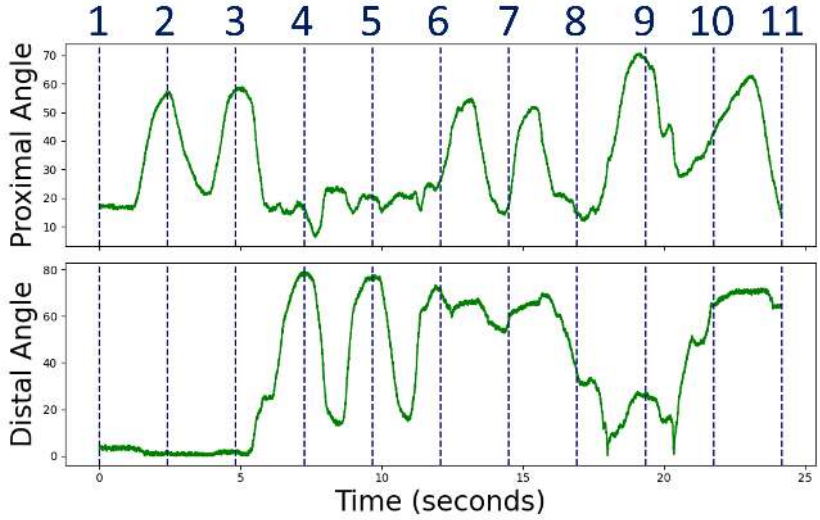
Yin, Z.-H., Huang, B., Qin, Y., Chen, Q., & Wang, X. (2023). Rotating without seeing: Towards in-hand dexterity through touch (cit. on p. 4).

Zhao, T. Z., Kumar, V., Levine, S., & Finn, C. (2023). Learning fine-grained bimanual manipulation with low-cost hardware (cit. on p. 4).

A Appendices

A.1 Angle Measurement using Markers and iPhone





A.2 Oak -D S2: Camera module specifications

Camera Specs	Color camera	Stereo pair
Sensor	IMX378 (PY004 AF, PY052 FF)	OV9282 (PY003)
DFOV / HFOV / VFOV	81° / 69° / 55°	89° / 80° / 55°
Resolution	12MP (4056x3040)	1MP (1280x800)
Focus	AF: 8cm - ∞, FF: 50cm - ∞	FF: 19.6cm - ∞
Max Framerate	60 FPS	120 FPS
F-number	1.8 ±5%	2.0 ±5%
Lens size	1/2.3 inch	1/4 inch
Effective Focal Length	4.81mm	2.35mm
Distortion	< 1% AF, < 1.5% FF	< 1%
Pixel size	1.55µm x 1.55µm	3µm x 3µm

A.3 Intrinsic Matrix of iPhone 13 Pro

$$\text{Intrinsic Matrix } K_{iPhone} = \begin{bmatrix} 1015.97 & 0 & 365.58 \\ 0 & 1014.80 & 644.46 \\ 0 & 0 & 1 \end{bmatrix}$$

A.4 Intrinsic Matrix of Oak -D S2 MonoCamera

$$\text{Intrinsic Matrix } K_{MonoCamera} = \begin{bmatrix} 818.97 & 0 & 663.94 \\ 0 & 818.57 & 382.029 \\ 0 & 0 & 1 \end{bmatrix}$$

A.5 Oak -D S2 120 FPS Videofeed Publisher Script

```
# Create pipeline
pipeline = dai.Pipeline()

# Define source and output
camMono = pipeline.create(dai.node.MonoCamera)
xoutVideo = pipeline.create(dai.node.XLinkOut)

xoutVideo.setStreamName("video")

# Properties
camMono.setResolution(dai.MonoCameraProperties.SensorResolution.THE_720_P)
camMono.setBoardSocket(dai.CameraBoardSocket.CAM_B)

xoutVideo.input.setBlocking(False)
xoutVideo.input.setQueueSize(1)

camMono.setFps(120)
# Linking
camMono.out.link(xoutVideo.input)

# Initialize ROS
rospy.init_node('frame_recorder', anonymous=True)
image_pub = rospy.Publisher('image_raw', Image, queue_size=10)
bridge = CvBridge()

# Connect to device and start pipeline
with dai.Device(pipeline) as device:

    video = device.getOutputQueue(name="video", maxSize=1, blocking=False)

    while not rospy.is_shutdown():
        videoIn = video.get()
        frame = videoIn.getData().reshape((videoIn.getHeight(), videoIn.getWidth()))
        frame = cv2.normalize(frame, None, 0, 255, cv2.NORM_MINMAX)
        frame = frame.astype(np.uint8)

        # Publish frame as ROS Image message
        ros_image = bridge.cv2_to_imgmsg(frame, encoding="mono8")
        image_pub.publish(ros_image)
```