FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Glove prototype for feature extraction applied to learning by demonstration purposes

Tiago João de Oliveira Leite Gonçalves Cerqueira

FOR JURY EVALUATION

MASTER IN ELECTRICAL AND COMPUTER ENGINEERING

Supervisor: Vítor Hugo Machado Oliveira Pinto

Co-supervisor: Gil Manuel Magalhães de Andrade Gonçalves

July 1, 2022

Abstract

This document is mainly focused on sensoring gloves, aimed to improve human-computer interface (HCI) and suitable for teleoperation tasks. This presents the main technological approaches, some theoretical aspects and the general context of possible solutions.

As technology progresses, more and more human-machine interfaces become relevant, and the need for reliable and accurate solutions increases to control robots in teleoperation systems.

The present work focus on a sensorial glove, capable of acquiring human hand motion and estimate its pose. The system needs to be accurate, consistent and relatively invariant to environmental conditions. The data acquired can then be used to control a robotic hand manipulator. The presented solution in this dissertation features an array of twelve IMUs to track orientation where the sensors are attached to an ordinary consumer glove.

This work also focus on the sensor fusion algorithms of IMUs and implementations, diving deeper in the algebraic quaternion algorithm (AQUA) because of its modularity and intuitive implementation. The human hand model is proposed, explaining its advantages and as such its limitations proposing ways to simplify the model. The distal phalanx dependency characteristic is an example, a way to estimate the distal phalanges orientation without directly measuring it is featured in the glove. Because calibration is so important in gyroscope performance, the online and offline calibration data was analyzed pointing its challenges and improvements. To better visualize the model and sensors it includes a simulation environment solution in *Unity* leaving enough room to grow and perfect it.

Acknowledgments

I would like to thank my supervisors Professor Vítor Pinto and Professor Gil Gonçalves for the, much needed, support and counselling of this work. For the quick productive meetings and the upfront readiness to source the material I am grateful, without it, this wouldn't be possible.

To my long-term relationship with Faculdade de Engenharia da Universidade do Porto and my department of Electrical and Computer Engineering, I am pleased to have studied here. Also a special word to my colleagues that walked by my side and shared the same struggles as me: Apes together strong.

A warm thanks to my family and friends that carried me not only in this chapter but long before, that helped me directly and indirectly in this achievement. I'm forever grateful for their support, they've watched me grow as a person and engineer and played an important role in my development throughout these long years.

An honourable mention to my ex assistant teacher of Programming, Luís Paulo, that expanded my knowledge beyond the required for the tests which pushed me to learn more and more, the passion I developed about computers is credited to him. Thanks for the time spent always on-call when I was struggling.

To my closest and dearest Francisca, for the patience and support in good and hard times, for playing the role of my right-hand and for sharing a great part of the memories and moments throughout these years, I feel very lucky and thankful.

Tiago Cerqueira

Contents

List of Figures

Abbreviations and Symbols

HCI	Human computer interface
HMT	Human motion tracking
CV	Computer vision
WS	Wearable sensor
IMU	Inertial measurement unit
IMMU	Inertial and magnetic measurement unit
MARG	Magnetic, Angular Rate, Gravity
VR	Virtual reality
MEMS	Micro Electrical Mechanical Systems
EKF	Extended Kalman filter
UAV	Unmanned aerial vehicle
GA	Generic algorithm
PF	Particle filter
BIPM	International bureau of weights and measures
TOA	Time of arrival
UWB	Ultra-wideband
BAN	Body area network
SPI	Serial peripheral interface
I^2C	Inter-integrated circuit
AHRS	Attitude heading reference system
DOF	Degrees of freedom

Chapter 1

Introduction

1.1 Context

A robotic hand that is attached to a multiple-degrees of freedom arm is considered useful in performing complicated tasks in various environments, e.g. in nuclear power plants, in space, and at the bottom of the sea. One growing application of robot hand manipulation used in everyday life is human-assistive technology. Human intelligence is more reliable than computers when it comes to make decision and take control on unstructured environments because of its capability to adapt, so there's a growing range of applications to robots human-assisted for those types of situations. Thus, robot teleoperation is necessary in this situation especially when objects are unfamiliar and unknown [1]. Commonly used human-robot interfaces examples, like joysticks [2]–[3] [4], dials and robot replicas This can used for risky assembly line operations, bomb disarming robots or possibly in the surgical medicine field. But the data glove can serve its own purpose for other applications such as sign language translation where it gives input to a translation system or to virtual or augmented reality environment.

Human motion tracking (HMT) systems could be divided into two categories: computer vision (CV)-based and wearable sensor (WS)-based [5]. The advantages of CV-based technology come from deploying web-depth cameras to monitor human activities, and has been applied in real world applications, such as film shooting and security monitoring [5]. Technology companies have also put their efforts in researching and developing professional human motion tracking systems, namely Vicon and Optotrak. They perform with high accuracy in controlled scenarios, but they rely heavily on the environment and its conditions, such as lights and shooting angle. The nature of the method makes it only suitable for smaller and controllable situations [5].

CV-based solutions are used in the teleoperation system. It is fairly easy for visual systems to capture contours of the human in the majority of environments, but the captured images cannot provide enough information for tracking hands in three-dimensional space. It is because the derivation of the spatial position information can lead to multiple 2D-3D mapping solutions [1]. Furthermore, with a single source of capturing images it is possible that some fingers are occluded and results in a non-observable problem, leading to faulty or undefined calculation of the pose [1].

Nonetheless, it can be integrated in a multi-sensory system to complement other's sensors weaknesses, like the Inertial Measurement Unit (IMU) drift problem.

WS-based HMT systems, on the other hand, do not need to deploy devices in scenarios and don't rely as much in environment factors. These type of systems tend to be more reliable and accurate. Therefore, they can be more suitable for large-scale and dynamic applications. However, these systems also tend to be pricey and use specialized equipment.

Another popular way is developed by the data glove. This method adopts technologies such as inertial sensors, contacting electromagnetic tracking sensors, gloves instruments with angle sensors to track the operator hand or arm motion which completes the required task. Until now, different types of data gloves have been developed overtime, both commercial and prototype ones. Some of the commercial examples: 5^{th} Glove, the Cyber Glove, the HumanGlove, the P5Glove, and the ShapeHand [1].

1.2 Motivation

As of the technology available right now, there is a lot of implementations and concepts of WS-based data gloves but it's not established a "right" way of achieving it. In addition, teleoperation systems still have limitations and room for improvement specially in high precision tasks. Solutions developed for virtual reality (VR) applications are the most known but tend to be noisy because of the range of movement the application requires. Gloves developed for sign language interpretation don't need to be as accurate and are more focused of the interpretation of the pose estimation. So, there is a need of a accurate enough solution to make teleoperation of a hand manipulator desirable, and at its best so capable as a real human hand. There is a need for teleoperation in applications where it is dangerous for a human to manipulate but the accuracy and dexterity of a human hand is needed. It also can replace some single purpose teleoperation interfaces, like joysticks and buttons, because of its general and intuitive nature of using a hand.

1.3 Goals

The goal is to design a glove capable of capturing hand and finger motion and send the data to other processing unit and serve as input for other systems. A data glove (also known as sensorial glove) implementation relies heavily on the sensor choices to acquire the data. There isn't a single right configuration of sensors that will give the best results for every type of application. Factors like real-time usability, portability, cost, accuracy and precision need to be taken in account. For instance, this solution of data glove needs to be mainly accurate, reliable and operate in real-time because its primary focus is to source data to a robotic arm teleoperation system. It needs to be capable of gathering all sensor raw data efficiently to then process it in the best way to provide the accuracy needed and also be able to ensure a program cycle speed suitable for real-time applications with a latency that should not be noticeable to humans operators.

1.4 Document Structure

This dissertation starts by addressing the existent literature of this type of applications and proposed solutions, weighting its strengths and weaknesses. An introduction to IMUs is presented and explaining superficially how they work, its main problems and a variety of solutions to those problems. Afterwards the flex sensors are introduced as another solution for orientation estimation. Computer vision is addressed because there is already a lot of solutions taking that approach and why this application can't rely on that method. The importance of calibration is mentioned and the different methods to achieve it. Following the calibration, options of sensors are presented to complement IMUs in order to better estimate orientation by fusing multiple estimations. The way the data is output is also discussed presenting multiple approaches both at hardware and protocol layers.

The chapter of theoretical concepts describes the hand model adopted in the application, detailing about orientation representations and the interpolation of the distal phalanx and how the orientation is estimated from the data read by the IMU sensors. The quaternion-based complementary filter is explained as the sensor fusion algorithm used in the application as well as the adaptive gain technique. It also goes more in depth about the calibration method used for this application.

The prototype design chapter explains how the theoretical concepts were implemented and adapted. It mentions and justifies the component choices and exposes the requirements the components need to meet. The iterations and the process of the tests about the communication protocols for IMUs is detailed and explained to understand the choice. Afterwards, regarding the software, the considerations and methods are exposed as well as the implementations of the concepts discussed earlier. It also explains how the glove interfaces with the simulator or other devices connected.

In the implementation and results chapter, the visualization is conceptualized to interpret the glove data. All the process and methods to design the hand in the simulator is detailed as well as the implementations to receive data and visualize. The chosen design of the glove (fabric) is evaluated and discussed. In the end it presents the results of the orientation estimation and the results of the online and offline calibration.

The document ends with the conclusions of the work and proposes improvements to future work.

Chapter 2

Literature Review

2.1 IMUs

An IMU is an electronic device that measures and reports the specific force, angular rate, and sometimes the orientation of the body, using an array sensors such as accelerometers, gyroscopes, and sometimes magnetometers (in case of Inertial and Magnetic Measurement Units (IMMUs)) [6], [7].

At the moment, IMUs are the main approach to WS-based HMT systems, such as gyroscopes and accelerometers. The basic principle is to measure the triaxial angular velocity and acceleration of the motion by these sensors, and obtain the orientation through integral operations. However, inertial sensors are known for the inevitable errors that accumulate over time [8], [9]. The drift error is the biggest challenge faced by IMU-based HMT systems [10], [11]. To improve HMT accuracy, the most common methods to tackle the drift error problem are as follows:

- The hardware aspect using high precision sensors to minimize the drift error, such as XSens and Invensense tracking units;
- The algorithm aspect enhances the system by using multi-sensor data fusion means.

The availability and advancements of Micro Electrical Mechanical Systems (MEMS) technology enables the inertial sensors integrated into single chips. It has made inertial sensors so small and low-power that they have already become a common practice in ambulatory motion analysis. For the time being, the magnetic sensors are used together with inertial sensors for accurate and drift free orientation estimation [12]–[14]. IMMU has been proved to be an approach accurate enough in estimating body segment orientations without external equipment [1]. It is easy to use and setup, non-intrusive and relatively cost effective. It demonstrates higher correlation and lower error estimation compared with a research-used visual motion capture system when recording the same motions. For this reason, the low-cost and small wearable inertial and magnetic sensors are becoming increasingly popular for the development of the data glove [1], [15].

As stated before, IMU sensors are most commonly implemented in MEMS for its compactness and low-power driven. This features make these chips closer to a seamless solution that feels like a normal glove and therefore improve mobility and agility of the hand.

An article [15] presenting the results of an analytical review of existing solutions and the this study [1] chose the *MPU*9250 which has the following key features:

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels;
- ±2/±4/±8/±16 g linear acceleration full scale;
- ±4/±8/±12/±16 gauss magnetic full scale;
- ±245/±500/±2000 dps angular rate full scale;
- 16-bit data output;
- SPI / I2C serial interfaces;
- Programmable interrupt generators.

Thanks to MEMS technology sensors can achieve great accuracy, speed and at a low cost. The features of the presented IMU sensor are typical of low-cost alternatives. Also, the *MPU*9250 sensor has already open-source libraries that should cover basic functionalities.

Given the above sensor packages, a sensor fusion algorithm is required to estimate orientation. Extended Kalman Filters (EKF) are frequently used in nonlinear estimation problems. The EKF can recursively estimate the system states from system measurements affected with Gaussian noises. The gyro has drift and the accelerometer has a Gaussian-like noise and that is what EKFs are mostly designed for [16].

- General Extended Kalman Filter

  Kalman filter is only suitable for linear systems, and requires that the observation equation is linear. Most of practical applications are nonlinear systems; therefore, the research of a nonlinear filter is very important But a lot of the applications the systems are nonlinear so the research for a nonlinear filter is also important. One of which is the Extended Kalman Filter that will be described [17]. The basic idea of EKF is to focus on the value of first-order nonlinear Taylor expansion around the status of the estimated, then transform the nonlinear system into a linear equation. EKF algorithm is commonly used in nonlinear filter systems, and the calculation is easy to be implemented. However, Taylor expansion belongs to linear process, so the system status and observation equations need to be close to linear and continuous, otherwise the EKF will not estimate well the true value. In addition, the status and measurement noise affect the filtering result [18]. The covariance matrix of the system status and observation noise do not change in the process of EKF. If the estimation of the covariance matrix of status and observation are not accurate enough, the error will accumulate and cause a divergence of the filter [19], [20].

Firstly, EKF is based on the first-order Taylor series expansion for nonlinear equation. Secondly, EKF assumes that the observations and status noise are independent in white noise processing, but the noise characteristics may not be coincident with the characteristics of the assumed white noise. Because first-order Taylor series expansion can change the observation and status noise, the assumption for noise will be also inconsistent with reality. Finally, for each EKF process the EKF needs to re-calculate the Jacobian matrix of the current time observation equation. Because the calculation of matrix is very complex, so it is difficult to solve the Jacobian matrix in some systems [19].

- Quaternion-Based Extended Kalman Filter

  Unit quaternions have many applications in state estimation problems because they are simple to work with. Crossbow Technology originally proposed a quaternion-based extended Kalman filter for the navigation IMU [5]. Developers of the UAV have used this approach on their specific platform. The system state variables include both the unit quaternion (q) and the gyro biases (b_p, b_q, b_r). The measurements or observation of the system are the accelerations: a_x, a_y, a_z, and the yaw angle derived from the magnetometer [16].

- Q-Method extended Kalman Filter

  The proposed method smoothly incorporates the nonlinear estimation of the attitude quaternion with the Davenport's q-method and the estimation of the nonattitude states through an extended Kalman Filter [21].

Other methods of attitude estimation in Attitude Heading Reference Systems (AHRS) [22] are very common:

- Algebraic Quaternion Algorithm

  Aside Extended Kalman filters approaches, there are other types of nonlinear filters such as complementary filters. They fuse the estimation of, for example, the gyroscope and the accelerometer with a confident gain approach where the trust in each sensor can be adjusted. The AQUA filter estimates and fuses the data in quaternion representation [23], [24]. It consists of estimating the orientation from the angular rate and correct that estimation with accelerometer and magnetometer readings. It computes two quaternions, the "tilt" quaternion and the "heading" quaternion from the magnetometer [25].

- Fast Accelerometer-Magnetometer Combination

  Accelerometer-Magnetometer combination (AMC) is the most typical low-cost sensor system. It integrates the local gravity and the Earth's magnetic field together to form a full-attitude estimation system [26]. The matrix operations are the main focus of attention in this method which are analytically simplified, where the accuracy is maintained, while the time consumption is reduced, hence the name Fast Accelerometer-Magnetometer Combination (FAMC) [27], [28].

- Madwick orientation filter

  Sebastian Madgwick introduced the sensor fusion algorithm Madwick named after him [29] in 2010. The proposed method is applicable to IMUs consisting of a tri-axial gyroscope and accelerometer, and MARG sensor arrays that also include tri-axial magnetometer. The advantages of this filter are: computationally inexpensive, requiring 109 (IMU) or 277 (MARG) arithmetic operations each filter update, efficient at low sampling rates and better performance than Kalman-based algorithms.

- Mahony orientation filter

  This method proposed by Robert Mahony *et al.* [30] is formulated as a deterministic kinematic observer on the Special Orthogonal group SO(3) [31] based on the instantaneous attitude and and angular velocity measurements. By taking advantage of the geometry of the special orthogonal group a related observer, the passive complementary filter, decouples the gyroscope measurements from the reconstructed attitude in the observer inputs [30]. The algorithm also estimates the gyroscope bias on-line with gyroscope and accelerometer outputs and an Explicit Complementary Filter [30].

2.2 Flex sensor

A flex or bend sensor is a sensor that measures the amount of deflection or bending. Usually, the sensor is completely stuck to the surface, and resistance of sensor element is varied by bending the surface, since the resistance is directly proportional to the amount of bend [32]. These sensors can be attached to the glove, using the same principle as the strain gauges (changing their resistance on the bending like in figure 2.1), but they have large resistance differences. For example, they can present 13-KΩ resistance on zero degrees bending, and 35-KΩ on 90°. This produces less noise when the signal is fed into the data acquisition system [33]. Consequently, the bending of a finger will deflect the surface of the glove and cause a resistance variance that can be measured to infer finger pose. However this solution assumes little or non lateral movement of the fingers which can be limiting.

Figure 2.1: Flex sensor based data glove [33].

2.3 Computer Vision

Visual hand gesture based interfaces have been used for navigation of virtual environments (VE) and control of a robot arm in robotic systems. Vision-based methods have the advantage of being easy to integrate, usually with basic equipment. This application also has the advantage of being intuitive and easy to control. Keep in mind, hand movement has a wide variety and lot of expression with complex finger motions. This paper [34] proposes a learning integrated with optimization approach introducing pose estimation and a motion-tracking technique with genetic algorithm (GA)-based particle filter (PF). It is described as a solution of high-dimensional and multi-modal search problem. Test results show promising capabilities of such a hand motion capture system, especially dealing with high dimensional search problem [34].

Other popular methods are marker-based motion capture. In essence, it is through established fixed and known references, placed on a body, that makes it possible to, as a result, acquire the data visually. These references make the mapping part more reliable and translatable. In this arcticle [35] was designed a 20 to 30 markers model and attached on the hand. One of the challenges of this solution is how to build temporal correspondences for all visible markers over time. Automatic marker labeling, particularly for hand capture, remains challenging while manually annotating markers over time is not practical and is also error prone [35].

Vision-based motion tracking methods due to significant finger and hand occlusion are often not capable of acquiring high-fidelity hand motion data consistently. However, there's solutions to minimize to chances of occlusion by sampling more than one angle of the scenario and then fuse the captured data.

2.4 Calibration

In order to guarantee sensor accuracy, calibration is needed. In case of the IMUs and IMMUs is possible that in mid-operation a calibration is required because scale of accelerometers and

magnetometers, and the bias of gyroscopes. According to the International Bureau of Weights and Measures (BIPM) calibration is defined as: "Operation that, under specified conditions, in a first step, establishes a relation between the quantity values with measurement uncertainties provided by measurement standards and corresponding indications with associated measurement uncertainties (of the calibrated instrument or secondary standard) and, in a second step, uses this information to establish a relation for obtaining a measurement result from an indication" [36].

The typical calibration principle is to subject the inertial sensors to a known angular velocity and linear acceleration and choose the calibration parameters such that the observed sensor output becomes as likely as possible. This method usually needs complex equipment, such as a turntable. A simplified calibration can be used, which only consider the main calibration parameters including the scale of accelerometers and magnetometers, and the bias of gyroscopes. There are two procedures of calibration, the online and the offline procedure. The online calibration is implemented to compensate the gyro bias or gyro drift. The ellipsoid fitting method is implemented by rotating the IMMUs in different orientations to estimate the magnetometer parameters. The offline calibration consists in determining the bias and scale of the accelerometers and magnetometers. When the calibration is executed, the measurements of these sensors are more reliable and therefore a better estimation is achieved [1].

Finding the gyroscope's offsets in offline calibration is not accurate by simply sampling the static gyroscope and computing the mean. Because temperature correlates with the gyroscope offset, the bias estimation in the moment of the offline calibration will be different than during usage due to temperature differences. The relation between temperature and the offset can be considered linear so one approach to estimate the offset is to profile the offset at different temperatures offline and find the temperature-offset coefficients. Then during usage the offset is estimated by reading the temperature and extrapolate with the linear curve. This study [37] proposed placing the sensors in a freezer to a cold temperature (4ºC) and sample the readings until it matches room temperature.

2.5 Orientation and Position Estimation

2.5.1 IMU hand configuration

To answer the questions such as where the sensors will be placed, how many sensors are needed and what grade of sensors is suitable for the application, all options must be weighted. The main features in this human-machine interface motion capture system is performance and accuracy, therefore the sensor choices and configuration must accommodate and maximize those two parameters of the system [38]. A possible solution is to use inertial and magnetic measurement units placed on the hand and finger segments. For example, placing an IMU in each phalanx and one reference in the back of the hand to a total of sixteen sensors to detect the movements for of the fingers and at least one on the back of the hand to detect the wrist movement, hand pose and serve as reference for the fingers [1] like pictured in figure 2.2.

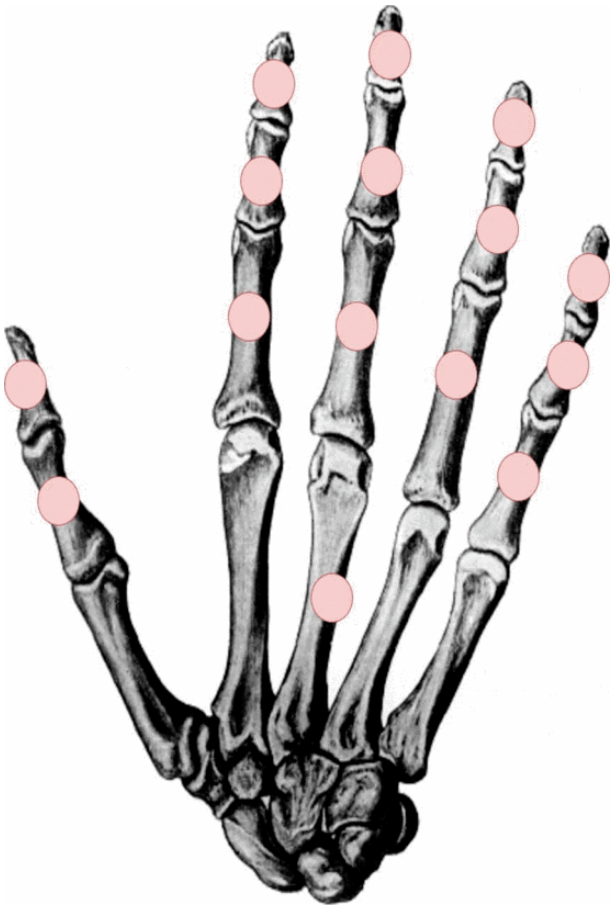

Figure 2.2: All phalanges configuration [15].

This sensor configuration (figure 2.2) minimizes mistakes in determining the orientation of the fingers and allows the organization of an optimal human-machine interface for the control of bionic prostheses, since this number and arrangement of the sensors makes it easy to determine the length of the operator's fingers as well as the angles between the phalanxes. This solution (figure 2.3) ensures maximum accuracy and the system gets rid of some ambiguities [15]. In this configuration, there is no need to integrate any positioning system because every angle of the fingers is measured and forwards kinematics calculates the pose of the hand.

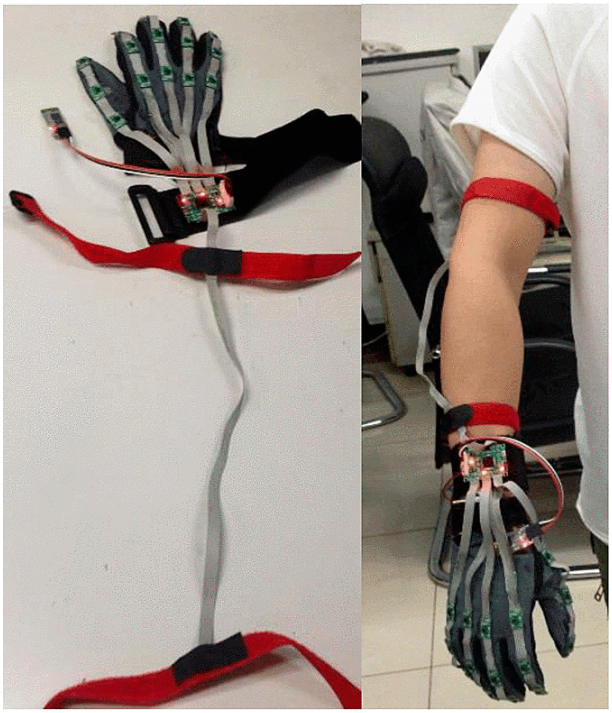

Figure 2.3: IMU-based data glove and arm [1].

Another way to configure the sensors can be achieved by placing only sensors at the tips of the fingers and one reference in the back of the hand. In this configuration the orientation alone isn't enough because there's no measurements of adjacent phalanges and the fingertips orientation doesn't define a single pose which is why a tracking solution is needed to to solve ambiguity. With the accelerometer data is technically possible to estimate the position of the fingertip in 3D space in reference to the IMU at the back of the hand by integrating twice. This solution has a drift problem too and even worse than the IMU orientation estimation because the error grows in a quadratic manner as it gets the first integration error and then integrate again on top of the first error. This way, errors will also accumulate and deviate from the true position of the fingers very quickly making this solution not good even with the best sensors. To tackle this other position tracking solutions are needed such as time of arrival (TOA) or computer vision (CV). These solutions will be later addressed in this section.

2.5.2 IMU-based Alternative

For this solution to estimate the position and orientation of the fingers the configuration used must be the sixteen total IMUs placed in each phalanx because the angle of each finger joint must be known and no other sensors are used. Based on the measured 3D angular velocity, acceleration and magnetic field of one single IMU, it is possible to stably estimate its orientation with respect to a global (gravity and magnetic north aligned) coordinate system. For this, angular velocity is integrated to obtain absolute orientation. Due to noise and offsets present in the sensor measurements, the resulting orientation exhibits strong drift. Accelerometer and magnetometer

measurements are therefore used to counteract this drift. By assuming negligible body acceleration, the accelerometers can be modeled as inclinometers (measuring only acceleration due to gravity) providing absolute information about two angles, i.e. pitch and roll. By assuming local static magnetic field homogeneous throughout the whole arm, the magnetic field measurements, projected into the horizontal plane, provide absolute information about the heading direction. By fusing all the measurements, an almost drift-free and long-term stable orientation estimate can be obtained. The above assumptions and principles can be formalized in a nonlinear state-space model for estimating the IMU orientation and kinematics within an extended Kalman filter. After the orientations of the each hand segment are determined, the positions can be derived using forward kinematics. Hence, the stable orientations and positions of human hand are estimated by the data glove [1].

However, this method cannot fundamentally solve the drift problem. High precision hardware used by wearable tracking systems is usually very expensive. In the aspect of algorithms, filtering methods, such as Kalman and particle filter, can somehow slow down the error accumulation process, but cannot eliminate it completely. Thus, the inevitable drift problem of inertial sensors has become a crucial constraint for wearable human motion tracking systems, which limits its use in long term or large space applications [39].

2.5.3 IMU plus TOA Alternative

In essence, human motion tracking (HMT) can be viewed as a local multi-target real-time high-precision three-dimensional positioning problem. Ultra-Wideband (UWB)-based time-of-arrival (TOA) ranging does not have the drawback of accumulative errors, compared with inertial-sensor-based HMT systems. The size of TOA chip is small enough to be integrated into wearable devices. Thereinto, IMU/TOA fusion is an effective way to overcome the accumulative errors of drifting problem faced by solo IMU method [39]. Some achievements on IMU/TOA fusion have been reported in many studies. However, state-of-the-art studies, mainly face the following two drawbacks:

- Requirements of fixed external anchors: They need to be deployed in certain scenarios. for example, Zihajehzadeh *et al.* [40] introduced a magnetometer-free algorithm for human lower-body motion capture by fusing inertial sensors and an UWB localization system. However, is not suitable for body area network (BAN) applications.

- State-of-the-art studies seldom concentrate on the fundamental limits of IMU/TOA fusion methods, and the error correction effects are not satisfying. For example, Nilsson *et al.* [41] proposed a cooperative localization method by fusion of dual foot-mounted inertial sensors and inter-agent UWB ranging, but the experiment results show that, compared with the performance lower bounds [42], there is still a lot of room for improvement.

In this paper [39] is proposed a IMU/TOA fusion-based human motion tracking alternative method. Time of Arrival (TOA) is considered to compensate the drift error caused by IMU. The

results obtained by simulation show a higher accuracy and little drift problem compared to traditional IMU/TOA tracking approaches. In a practical application, this IMU/TOA fusion method also shows performance advantages when compared with state-of-the-art methods. On top of that, it does not need external anchors, thus it gives more flexibility and resilience to scenarios and it is more suitable for wearable motion tracking applications [39].

2.5.4 IMU plus CV Alternative

Methods to estimate position and orientation using IMU sensors aim to tackle the drift error problem and surely the technology has matured and everyone carrying a phone most likely carries an IMU sensor that tracks orientation to a certain accuracy, but some just manage to drift slower and require recalibration. On the other hand, vision-based capture does not have this temporal problem. The two methods can complement each other, the IMU sensors lack consistency but are accurate, while vision-based are consistent but computationally intensive and not reliable enough, but through sensor fusion can compensate the IMU sensor bias.

2.6 Output Data

The way the data captured by the glove can be transmitted as output, can vary both in protocol and physically. The method that suits best the application needs to take in consideration the speed and data integrity, but as communication technologies matured this problem became less relevant and mainstream solutions are well documented and available. It is possible to make the glove transmit the data via Bluetooth or other wireless protocols to make the glove prototype more portable and maneuverable instead of a wired transmission but the wired transmission has the advantage of having faster solutions, is cheaper and usually more reliable.

The output of an IMU is usually transferred via SPI or I^2C, both well established communication protocols. Today, at the low end of the communication protocols we find the inter-integrated circuit (I^2C) and the serial peripheral interface (SPI) protocols. For communications between integrated circuits for slow communication with on-board peripherals both protocols are well suited, they coexist in modern digital electronics systems, and they most likely will continue to compete in the future, as both I^2C and SPI quite complement each other for this kind of communication.

Chapter 3

Theoretical Concepts

3.1 Hand model

A simple skeletal dexterous hand model was used to understand the data that the glove outputs and later visualize in a simulator. It consists in a fairly accurate anatomical model but with some constraints and assumptions. Of the five fingers, four are very similar and the thumb is slightly different, the index, middle, ring and pinky fingers consist of four segments connected by joints.

Starting near the wrist, the metacarpus is the largest and the one visible in the back of our hands, it has little to no movement therefore it is assumed not flexible and because the four of them form, it is rigid and flat in this model. Connected to the metacarpus start the phalanges with the proximal phalanx and is the only phalanx capable of moving laterally, the intermediate and distal phalanges only rotate in one axis. All phalanx movement will be taken into account and in the model. However the sensors used are capable of estimating all kind of rotation and due to errors they can estimate physically impossible finger orientations, so some assumptions can be made about these physical limitations and improve the estimation (will be later discussed in section 3.2). The thumb behaves like three phalanges with the same properties and capabilities mentioned before but anatomically is composed by a metacarpus, proximal phalanx and distal phalanx, so it differs from the other fingers by lacking a intermediate phalanx and the metacarpus behaves like a proximal phalanx.

Not always we can move the muscles independently, one common example is trying to move just one finger like the ring finger and realize the pinky finger also moves, this study [43] reveals not only the fact but measures how much they move independently. Grasping this concept one might suggest the same happens to our distal phalanx (tip of the finger). Although some individuals are capable of moving the distal phalanx independently, this work will assume it's not common in normal daily hand activity. The independence of the distal phalanx of the index, middle, ring and pinky fingers will be considered negligible therefore their orientation estimation will be a function of the proximal and intermediate phalanges.

The model shown in figure 3.1 is very similar to the one proposed in paper [37] with some minor differences. In the paper it is also used 2 IMUs per finger but instead tracks the proximal

and distal interpolating the intermediate [37] while in this application the interpolated phalanx is the distal. In the paper the model doesn't consider the metacarpal thumb movement while in this application the movement of all 3 phalanges is tracked.

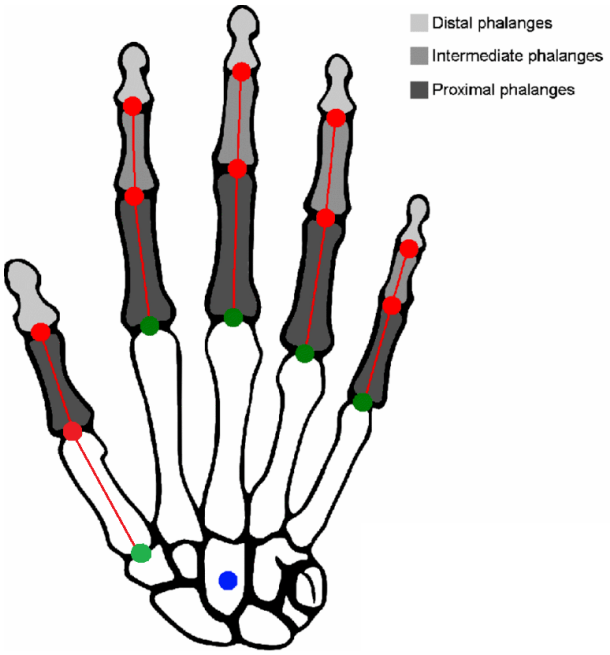

Figure 3.1: Proposed hand model (adapted from [37]).

3.2 Orientation estimation

In order to estimate orientation in 3D space is very common to use IMUs because they are high portable devices increasingly used in human motion capture systems [44]. They offer a reference (gravity) given by the accelerometer and a measurement of rotation by the gyroscope. It is enough to initialize and estimate attitude commonly known by pitch and roll but not heading or yaw, to obtain the heading the IMU is usually packed with a magnotemeter that it can be used as a compass. In this application, the heading is not so important and magnetometers are the least reliable of the three sensors. Ideally, the sensors would give perfect measurements but that is not the case and its implications will be addressed in this future section 3.2.1.

For instance, assuming that the IMUs measures are perfect and the orientation estimation is computed from that (extracting orientation from the IMUs data later addressed in section 3.3) there is many ways to represent an orientation, namely quaternions, rotation matrices and the many types of Euler angles [45].

Leonard Euler (1707–1783) proved that given a known sequence of axis to rotate, only 3 angles are needed to define a rotation between any two independent orthonormal coordinate frames with a common origin [45]. Named after Leonard Euler (1707–1783), Euler angles are the simplest to grasp because it consists of rotating around 3 axes by 3 angles. The representation shown in figure 3.2 is very common for simple quad-copters or aviation also known for roll, pitch and

yaw angles which intuitively one can comprehend and interpret those numbers. However, it is susceptible to gimbal lock where an axis overlaps another with the rotation and loses one degree of freedom because now angles rotate in the same axis [46].

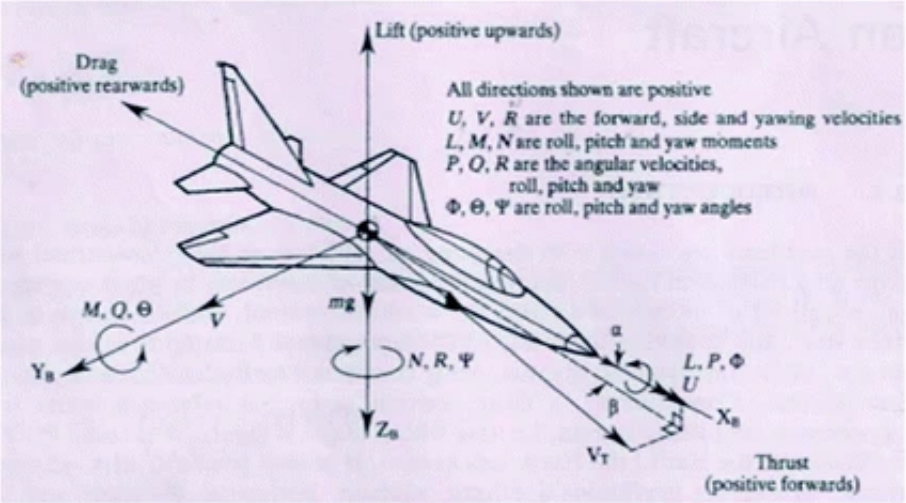

Figure 3.2: Roll, pitch and yaw axis commonly used in aircrafts [47]

Rotation matrices are very flexible and well established in all sort of transformations mainly due to the convenience of matrix algebra operations but are not so intuitive just by looking at the a matrix defining the rotation [45]. So, in this work, quaternions were the best choice, they are known for the fast and easy computation, compact memory use and to avoid singularities such as gimbal lock in Euler angles. In addition, SLERP (spherical linear interpolation) are very easy to compute from two quaternions [24].

Quaternions are an extension of complex numbers but they add not one but two imaginary parts, composed by a real part, and 3 imaginary parts with a distinct multiplication ruleset. The same way complex numbers describe rotations in 2D space, quaternions inherit that to the 3D space. It can be represented as 4 real coefficient numbers of (w, i, j, k) or (w, x, y, z). This is common because quaternions are very close to angle-axis representation, as the name implies describes a rotation in the axis (e_x, e_y, e_z) by an angle α, the quaternion is formed as:

$$\mathbf{q} = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\frac{\alpha}{2} \\ e_x \sin\frac{\alpha}{2} \\ e_y \sin\frac{\alpha}{2} \\ e_z \sin\frac{\alpha}{2} \end{bmatrix} \tag{3.1}$$

and to rotate and change frames of reference quaternion multiplication is similar to rotation matrices. We can multiply quaternions **p** and **q** as:

$$\mathbf{pq} = \begin{bmatrix} p_w q_w - p_x q_x - p_y q_y - p_z q_z \\ p_w q_x + p_x q_w + p_y q_z - p_z q_y \\ p_w q_y - p_x q_z + p_y q_w + p_z q_x \\ p_w q_z + p_x q_y - p_y q_x + p_z q_w \end{bmatrix} \tag{3.2}$$

A finger can be thought of 3 links that are the phalanges and 3 joints that are the articulations between them. Usually, in robotics, the transformation matrices are referenced to the last link but in this case using IMUs, the orientation estimated is in the global frame of reference. The relative orientation is obtained by finding the quaternion that describes the rotation between two quaternions, in other words the quaternion that transforms for one frame of reference to another.

3.2.1 Interpolate distal phalanges

As discussed in section 3.1 it can be assumed that the distal phalanges are not independent so they are mostly a function of the other phalanges. This work will not go in depth and find the most precise model but to introduce this concept and prove it's a good approximation. This concept can be useful to save sensors that most of the time would be redundant, lift some processing load and cost.

Firstly, it is evaluated how the distal phalanges depend on and empirically reach the conclusion that they depend on the angle between the proximal and intermediate phalanges. A linear approximation is done in the relation between the angle of the proximal and intermediate phalanges and the angle of the distal phalanx. To find the factor between those angles, the straight finger angles and the fully curled finger angles are measured as shown:

Figure 3.3: Straight and curled finger angles.

Obtaining these results in the table below:

Configuration	Between	Distal
Straight	0	0
Curled	115	65

So the function between the angle of the proximal and intermediate phalanges (α) and the angle of the distal phalanx (β) is described as:

$$f(\alpha) = \beta = \begin{cases} \frac{65}{115}\alpha & ,\alpha \geq 0 \\ 0 & ,\alpha < 0 \end{cases} \tag{3.3}$$

3.3 IMU

As mentioned before, there are a lot of approaches to estimate orientation and improve performance and reliability of these sensors. Although the gyroscope is very accurate for a short period, it lacks precision in the long-term due to a bias in the angular velocity readings as we integrate, the error accumulates. In the other hand IMUs also pack an accelerometer which isn't good for short duration, because is very noisy but perform well in long-term because it doesn't drift. So the two together are meant to complement each other and take advantage of both sensor strengths. Some IMUs also pack a magnetometer to correct for heading in relation to earth north pole but it will not be taken advantage of in this application because it is very susceptible to environmental factors that distort the magnetic field and even more noticeable in industrial environments. So, most algorithms and approaches aim to tackle this sensor fusion problem to get the best of both sensors.

3.3.1 Gyroscope

Gyroscopes measure angular rate usually in radians per second. This type of sensors are arranged in 3 axes to measure angular rate in 3D space. To track our orientation we need to integrate the angular rate but with the gyroscope there is no information about the absolute orientation in the global frame, so it is relative to the orientation when it started measuring and it is not limited by any starting orientation.

The main advantages of the gyroscopes is the precision and the fast response time. However, the sensors carry a bias in the readings and because orientation is tracked by integrating these readings, this bias accumulates an ever-growing error in the orientation. To minimize this effect a calibration is applied (section 3.5.3) which improves substantially the accuracy of the sensor however it is still not good enough to track orientation on its own.

The estimation of the orientation after reading the gyroscope measurements depends on the last orientation, the angular rate and the delta time between samples. It is computed as:

$$\begin{aligned}\mathbf{q}_\omega &= \begin{bmatrix} 1 & -\frac{\Delta t}{2}\omega_x & -\frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z \\ \frac{\Delta t}{2}\omega_x & 1 & \frac{\Delta t}{2}\omega_z & -\frac{\Delta t}{2}\omega_y \\ \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_z & 1 & \frac{\Delta t}{2}\omega_x \\ \frac{\Delta t}{2}\omega_z & \frac{\Delta t}{2}\omega_y & -\frac{\Delta t}{2}\omega_x & 1 \end{bmatrix} \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} \\ &= \begin{bmatrix} q_w - \frac{\Delta t}{2}\omega_x q_x - \frac{\Delta t}{2}\omega_y q_y - \frac{\Delta t}{2}\omega_z q_z \\ q_x + \frac{\Delta t}{2}\omega_x q_w - \frac{\Delta t}{2}\omega_y q_z + \frac{\Delta t}{2}\omega_z q_y \\ q_y + \frac{\Delta t}{2}\omega_x q_z + \frac{\Delta t}{2}\omega_y q_w - \frac{\Delta t}{2}\omega_z q_x \\ q_z - \frac{\Delta t}{2}\omega_x q_y + \frac{\Delta t}{2}\omega_y q_x + \frac{\Delta t}{2}\omega_z q_w \end{bmatrix}\end{aligned} \tag{3.4}$$

where $\mathbf{q} = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T$ is the last orientation and $\mathbf{q}_\omega$ is the predicted orientation. This is also known as the attitude propagation.

3.3.2 Accelerometer

The accelerometer measures acceleration including gravity. It consists of 3 measurements in 3 axes and define an acceleration vector. Intuitively, a standing still object only has gravity acting on it and describe the acceleration vector pointing downwards with a 9,8 norm. However, that is not the correct answer, in the case of gravity it is interpreted as the surface below the accelerometer is pushing it upwards at 9,8 m/s² counter-acting the force of gravity. Therefore, a free falling accelerometer, although being accelerated by gravity measures 0 acceleration in all axis.

So, the accelerometer does not measure coordinate acceleration, it actually measures proper acceleration and that is the acceleration in its own instantaneous rest frame, which is different from coordinate acceleration.

In this application, only gravity acceleration is relevant because ultimately the goal is to obtain an orientation estimation and other sources of acceleration will cause errors to that estimation. So assuming the accelerometer is only being affected by gravity it is possible to estimate pitch (θ) and roll (ϕ).

The accelerometer measurements should be filtered with a median or mean filter to reduce noise, these filters act as a low-pass filters and because of that they are not best suited for the gyroscope as it degrades its speed advantage and short duration precision.

It is computed from the filtered readings $\begin{bmatrix} a_x & a_y & a_z \end{bmatrix}$:

$$\begin{aligned}\theta &= \text{arctan2}(\mathbf{a}_y, \mathbf{a}_z) \\ \phi &= \text{arctan2}\big(-\mathbf{a}_x, \sqrt{\mathbf{a}_y^2 + \mathbf{a}_z^2}\big)\end{aligned} \tag{3.5}$$

and leave the yaw angle (ψ) as 0 because it cannot be computed without magnetometer. Roll, pitch and yaw representation is a common type of Euler angle representation, more specifically

Tait-Bryan angles. In navigation systems the order of rotation is usually ZYX (or yaw, pitch, roll) and we convert to quaternion form as:

$$\mathbf{q}_a = \begin{pmatrix} q_w \\ q_x \\ q_y \\ q_z \end{pmatrix} = \begin{pmatrix} \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) - \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) + \sin\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) \\ \cos\left(\frac{\phi}{2}\right)\cos\left(\frac{\theta}{2}\right)\sin\left(\frac{\psi}{2}\right) - \sin\left(\frac{\phi}{2}\right)\sin\left(\frac{\theta}{2}\right)\cos\left(\frac{\psi}{2}\right) \end{pmatrix} \tag{3.6}$$

This quaternion is most useful to initialize the glove orientation.

3.4 Euler angles complementary filter

Because IMUs track angular rate along 3 axes, tracking the Euler angles to describe rotation is as simple as increment the previous angle and sum how much it rotated given by the angular rate and delta time:

$$\mathbf{e}_t = \mathbf{e}_{t-1} + \omega\Delta t \tag{3.7}$$

This solution is very simple and is very easy to fuse with the accelerometer data with a complementary filter because for the Yaw angle no fusion is computed. With quaternions it is not so trivial and will addressed in section 3.5.

$$\begin{aligned} \mathbf{e}_x &= (1-\alpha)\mathbf{g}_x + \alpha\mathbf{a}_x \\ \mathbf{e}_y &= (1-\alpha)\mathbf{g}_y + \alpha\mathbf{a}_y \\ \mathbf{e}_z &= \mathbf{g}_z \end{aligned} \tag{3.8}$$

where $\mathbf{g}$ is the gyroscope angle prediction and $\mathbf{a}$ is the accelerometer angle prediction. The value of α is the trust in the estimation which will be explained deeper in section 3.5.

This orientation representation presents the gimbal problem as mentioned in section 3.2. For some applications this isn't as relevant, for example quadcopters (assuming they don't flip) because their working range avoids the gimbal lock. This method was tested and in fact, when the hand or finger were straight up or down, then the other two axis would lock and perform the rotation around the same axis. For this reason, Euler angles representation was not adopted.

3.5 Quaternion-based complementary filter

Complementary filters are the simplest methods for sensor fusion, they are simple to comprehend and are computationally inexpensive. They consist in a low-pass filter and a high-pass filter

for each filter with the same cut off frequency. It can also be interpreted as simple scalar gain (α) that describes the trust in each estimation.

Let us look at a gyroscope-accelerometer sensor fusion example, we compute a gyroscope estimation and an accelerometer estimation and we aim to estimate the true orientation of the body.

$$\mathbf{q} = (1-\alpha)\mathbf{q}_\omega + \alpha\mathbf{q}_a \tag{3.9}$$

Here quaternions and Linear Interpolation (LERP) are used because it is a good approximation for small differences between them, if a more accurate interpolation is desirable a Spherical Linear Interpolation (SLERP) can be used like so:

$$\widehat{\mathbf{q}} = \frac{\sin([1-\alpha]\Omega)}{\sin\Omega}\mathbf{q_w} + \frac{\sin(\alpha\Omega)}{\sin\Omega}\mathbf{q_a} \tag{3.10}$$

The gain α describes how close is from one of the two quaternions hence the trust in each estimation, for example, a gain of 0 means we fully trust gyroscope orientation prediction, and a gain of 0,5 means both estimations are equally trusted and the result of the estimation is a quaternion right in the middle of both gyroscope and accelerometer quaternions.

This method works very well, for example, for quadcopters that only aim to correct the tilt and ignore the heading or when the magnetometer is used and the heading can be computed (note that quadcopters can still take advantage of heading to fuse magnetometer and GPS data).

Because accelerometer lacks the heading information, a 0 degrees angle is assumed. But the gyroscope can track heading from its original orientation, thus presenting a challenge when fusing both. Gyroscope orientation prediction will track heading and accelerometer orientation prediction will always try to pull the heading back to 0 degrees. This can be fine for some applications but for the data glove this is not intended.

To get around this problem, a correction-based method can be used. It works by taking orientation prediction of the gyroscope and then correct that prediction with delta quaternions. This method separates the correction of the attitude and correction of the heading, from the accelerometer and the magnetometer respectively.

$$\mathbf{q} = \mathbf{q}_\omega \Delta\mathbf{q}_{\text{acc}} \Delta\mathbf{q}_{\text{mag}} \tag{3.11}$$

but because there is no magnetometer measurements it is assumed that the magnetometer delta quaternion is a unit quaternion which means no correction.

3.5.1 Accelerometer-based correction

To find the accelerometer delta quaternion the measured gravity vector is rotated from the local frame to the global frame by the inverse predicted quaternion of the gyroscope:

$$\mathbf{R}({}^G_L\mathbf{q}_\omega)\,{}^L\mathbf{a} = {}^G\mathbf{g}_p \tag{3.12}$$

where ${}^{G}_{L}\mathbf{q}_{\omega}$ is the conjugate of gyroscope normalized predicted quaternion $\mathbf{q}_{\omega}$ and ${}^{G}\mathbf{g}_{p} = \begin{bmatrix} g_x & g_y & g_z \end{bmatrix}^{T}$ is the predicted gravity which has a small deviation from real gravity vector ${}^{G}\mathbf{g} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^{T}$. The accelerometer delta quaternion is computed from the rotation of ${}^{G}\mathbf{g}$ into ${}^{G}\mathbf{g}_{p}$:

$$\mathbf{R}(\Delta\mathbf{q}_{\text{acc}})\,{}^{G}\mathbf{g} = {}^{G}\mathbf{g}_{p} \tag{3.13}$$

The delta quaternion is used interpolated via LERP with identity quaternion $\mathbf{q}_{I} = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}$ by the gain α with value 0,005:

$$\overline{\Delta\mathbf{q}}_{\text{acc}} = (1-\alpha)\mathbf{q}_{I} + \alpha\Delta\mathbf{q}_{\text{acc}} \tag{3.14}$$

Normalizing the interpolated quaternion:

$$\widehat{\Delta\mathbf{q}}_{\text{acc}} = \frac{\overline{\Delta\mathbf{q}}_{\text{acc}}}{\|\overline{\Delta\mathbf{q}}_{\text{acc}}\|} \tag{3.15}$$

and finally correcting the orientation by multiplying the gyroscope predicted quaternion by the delta quaternion:

$$\mathbf{q} = \mathbf{q}_{\omega}\,\widehat{\Delta\mathbf{q}}_{\text{acc}} \tag{3.16}$$

This way the roll and pitch are corrected but the heading angle is not affected and totally estimated by the gyroscope. If there was magnetometer data available, a correction for the heading could be computed with the respective delta quaternion.

3.5.2 Adaptive gain

The current orientation estimation relies in the static condition assumed by the accelerometer measurements but, if the body is being accelerated by other forces than gravity, the estimations are evaluated using a false reference. However, the gyroscope measurements are not affected by linear acceleration, thus they can still be trusted to compute an accurate orientation estimation.

The current constant gain fusion algorithm struggles to accurately estimate orientation in this conditions because it is optimized to static conditions. To tackle this problem, an adaptive gain algorithm can be used.

First the relative error e_m is computed:

$$e_m = \frac{|\|a\| - g|}{g} \tag{3.17}$$

where a is the measured acceleration vector before normalization and $g = 9.81\,ms^{-2}$.

From the LERP and SLERP definitions, the filtering gain α becomes dependent on the magnitude error e_m through the gain factor function f:

$$\alpha = \overline{\alpha} f(e_m) \tag{3.18}$$

where $\overline{\alpha}$ is the optimal constant gain evaluated for static conditions and $f(e_m)$ is the gain factor that is a continuous function of the magnitude error.

$$f(e_m) = \begin{cases} 1 & \text{if } e_m \leq t_1 \\ \frac{t_2 - e_m}{t_1} & \text{if } t_1 < e_m < t_2 \\ 0 & \text{if } e_m \geq t_2 \end{cases} \tag{3.19}$$

When the magnitude error does not exceed a threshold t_1, the gain factor is equal to 1, this happens when the non-gravitational acceleration is not high enough to overcome the acceleration gravity. If the magnitude error is above the threshold, the gain factor decreases linearly until it reaches 0 at the second threshold t_2 and over.

The values for the thresholds t_1 and t_2 were 0.1 and 0.2 suggested empirically by article [24].

3.5.3 Calibration

A simple calibration was adopted in order to improve accuracy in the sensor readings. For the gyroscope, as noted before, it is known it carries a bias in the readings. This bias is fairly easy to approximate by sampling the gyroscope readings while the gyroscope is standing still, from those readings the mean is computed for each axis finding the offsets of the gyroscope. When reading the gyroscope angular rate the raw readings are subtracted by the offsets computed from the samples.

$$\omega = \omega_{raw} - \omega_{off} \tag{3.20}$$

No sensor is the same so this process needs to be done for every individual gyroscope. The initialization and calibration takes between 5 to 10 seconds for 200 to 300 samples so to improve this performance, an offline calibration was attempted. The idea is to calibrate the sensors once and then load those values for the respective sensors.

For the accelerometer calibration, multiple readings were sampled assuming the sensor is at rest and it should rest on the glove nearly flat. The mean of the samples was computed for each axis and computed the offsets, but in this case the offsets they are not all referenced to 0. The z-axis must be calibrated to -9,81 to account for gravity. This means that the moment the accelerometer was calibrated, it was assumed that its orientation was flat so it is important that during the calibration the hand is in a predefined known orientation, in this case, horizontal with straight fingers together and thumb in a natural position around 45 degrees. Essentially this method of calibrating consists of changing the accelerometer frame of reference, but the gyroscope frame of reference is not adjusted, so the readings are not well interpreted. So, a better approach is not to calibrate

the accelerometer but store the initial orientation computed from the accelerometer readings and correct the estimation before sending. The corrected estimation quaternion can be computed as:

$$\mathbf{q}_{corr} = \mathbf{q}_{ref}{}^{-1} * \mathbf{q}_{est} \tag{3.21}$$

Because only the pitch and roll are corrected by the magnetometer, over time the heading drifts from the true angle. One option is to restart, which will reset the position and calibrate every sensor again but the re-calibrate isn't strictly necessary because the drift usually is faster than the deviation in calibration. For convenience for the user of the glove, a reset button was added to reset the position from the accelerometer like when the orientation is first initialized (equation 3.6 readings without re-calibrating again).

Chapter 4

Prototype Design

4.1 Components

The IMUs have a wide range in performance and prices, but as an array of twelve of these sensors is being used, finding an affordable one is essential. Although top end IMUs improve our estimations and reduce drift, luckily the affordable ones do perform well enough for our application.

The sensor package used in this application was the *MPU*9250. The main features were its affordability, the 3,3V input option, the 9 degrees of freedom (DOF) (10 DOF with temperature) which means it packs a gyroscope, an accelerometer and a magnetometer to not limit our options, and SPI or I^2C as communication protocols. The characteristics of the MEMS sensors are very typical for the price range and full of other features that were not used. Another aspect of this sensor package is that widely used and there is a lot of support online and having an IMU library ready to use accelerates the development process.

For the brain of the glove the Teensy 4.0 was chosen mainly because it features 14 analog pins in case an array of flex sensors were implemented (which was not the case) and also enough digital pins to drive the sensors in case the SPI protocol is used, because each sensor requires a "chip select" pin. One major feature was the capability of single-clock float arithmetic, as the algorithms for the quaternions and the filtering require a lot of float operations. This board is full of features and very fast at an affordable price. It also features an ARM processor that fits the IMU library intended to use. Unfortunately, it did not feature any wireless solution useful for reducing cables and improving user experience.

A reset button was added as mention in section 3.5.3 to reset the position. This is the simplest design of a button, that when clicked pulls the digital pin *HIGH* like shown in the schematic 4.1. No debounce capacitor was used. Instead, a solution through software to interpret the click only once only if the signal rising-edge was stable for a minimum amount of time.

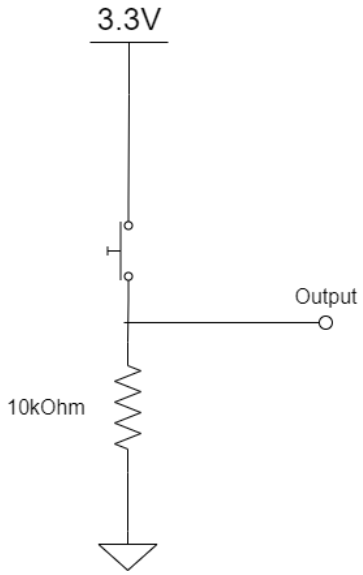

Figure 4.1: Button schematic.

4.1.1 Communication Protocol

At the beginning, the SPI protocol was chosen because the I^2C works by specifying the address of the slave it wants to communicate before sending the data, so to communicate with the different sensors, the sensors must have different addresses or they interfere with each other. However, it is not possible to freely change the I²C addresses of all twelve MPU9250 sensors, since there are only 2 addresses available that we can change by pulling a specific pin to *HIGH*. That is the main reason SPI seemed the better option and also has the advantage of reaching higher speeds.

All the IMUs were tested with I²C and SPI individually. A simple program was run for a while that repeatedly reads the sensor and prints through serial the values. The IMU sensor was rotated and accelerated in all axes to validate the readings. Afterwards, the program is left running for a while to validate stability. The I²C tests worked well and stably but the SPI tests within a minute, the IMUs with instability issues crashed very quickly.

The SPI was very inconsistent when testing all IMUs. Some IMUs struggled with the SPI clock speed, and even after lowering it, they were very susceptible to interferences, like hitting or moving the wires and would stop working. Even after selecting the most stable ones by testing one by one, after mounting all to the the same SPI bus working at the same time, some would stop working as well.

I^2C in the other hand, although it was only possible to test with a maximum of 2 sensors at the same time, revealed a stable connection even for the ones unstable for SPI, so if there is a way to overcome the I^2C addresses problem, it may stably work for twelve sensors. The first and most common solution found is an I^2C multiplexer. This piece of hardware functions as another I^2C slave with multiple isolated I^2C buses that switches which bus is active. The simple TCA9548A I^2C multiplexer was used because it has 8 channels. Every sensor can have 2 different addresses so up to 2 IMUs are connected per channel plus two in the main I^2C bus where the mulitplexer is

also connected. If the main I^{2}C bus is used only 5 channels are needed from the I^{2}C multiplexer but since there are channels to spare, they all get connected in the multiplexer's buses, like is simplified in the schematic 4.2.

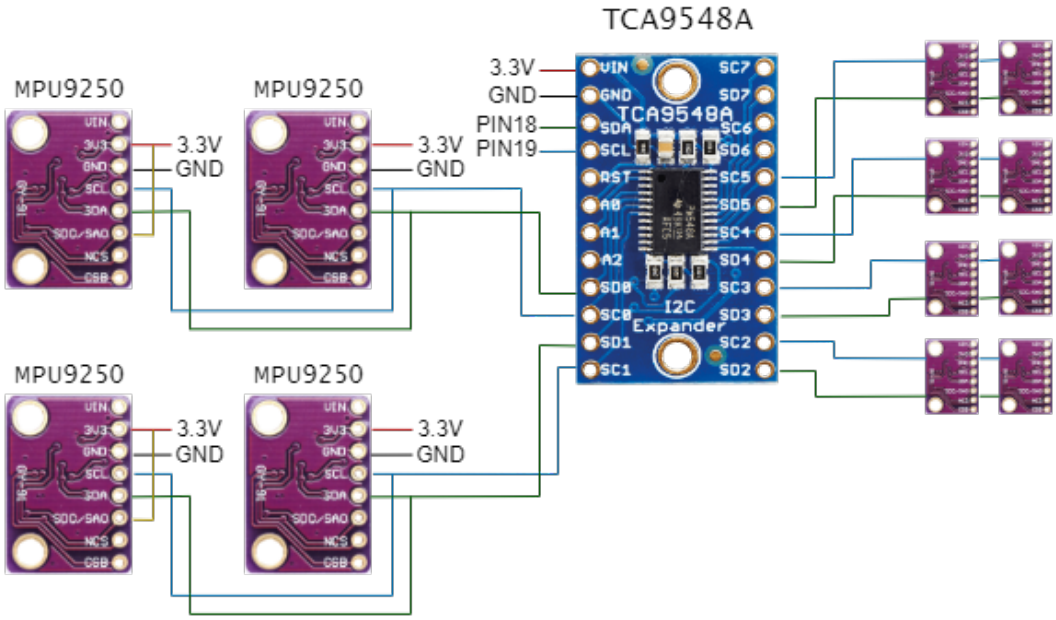

Figure 4.2: Multiplexer schematic.

The main I^{2}C bus pins *SDA* and *SCL* are connected to the respective pins in the multiplexer board and communicate like any other I^{2}C device in the bus. The default address is $0x70$ and can be modified by changing $A0$, $A1$ and $A2$ pins. A byte is sent to the target address with one bit set to 1 specifying the channel to activate.

It was tested by reading all sensors per loop with no delay starting with just two sensors in one channel and incrementally adding two in another channel. This worked very well with no setbacks and the goal to operate twelve IMUs simultaneously is considered realised.

4.1.2 Schematic

So the end result, simplified in the schematic 4.3, is the Teensy 4.0 microcontroller connected to the TCA9548A I^{2}C multiplexer by pins 18 and 19 to *SDA* and *SCL* respectively, this constitutes the main I^{2}C bus. The button is also connected to the microcontroller to digital pin 5 with a pull-up 10kOhm resistor. The TCA9548A I^{2}C multiplexer has 6 pairs of MPU9250 IMU sensors connected from channel 0 to channel 5. Each pair connects to the multiplexer by *SDA* and *SCL* pins and one the MPU9250 sensors per pair pulls the address pin to *HIGH* to change its address so this pin is connected to 3,3V. The microcontroller sources the power by connecting the ground and the 3,3V pin to a shared bus that all the components can connect to get powered.

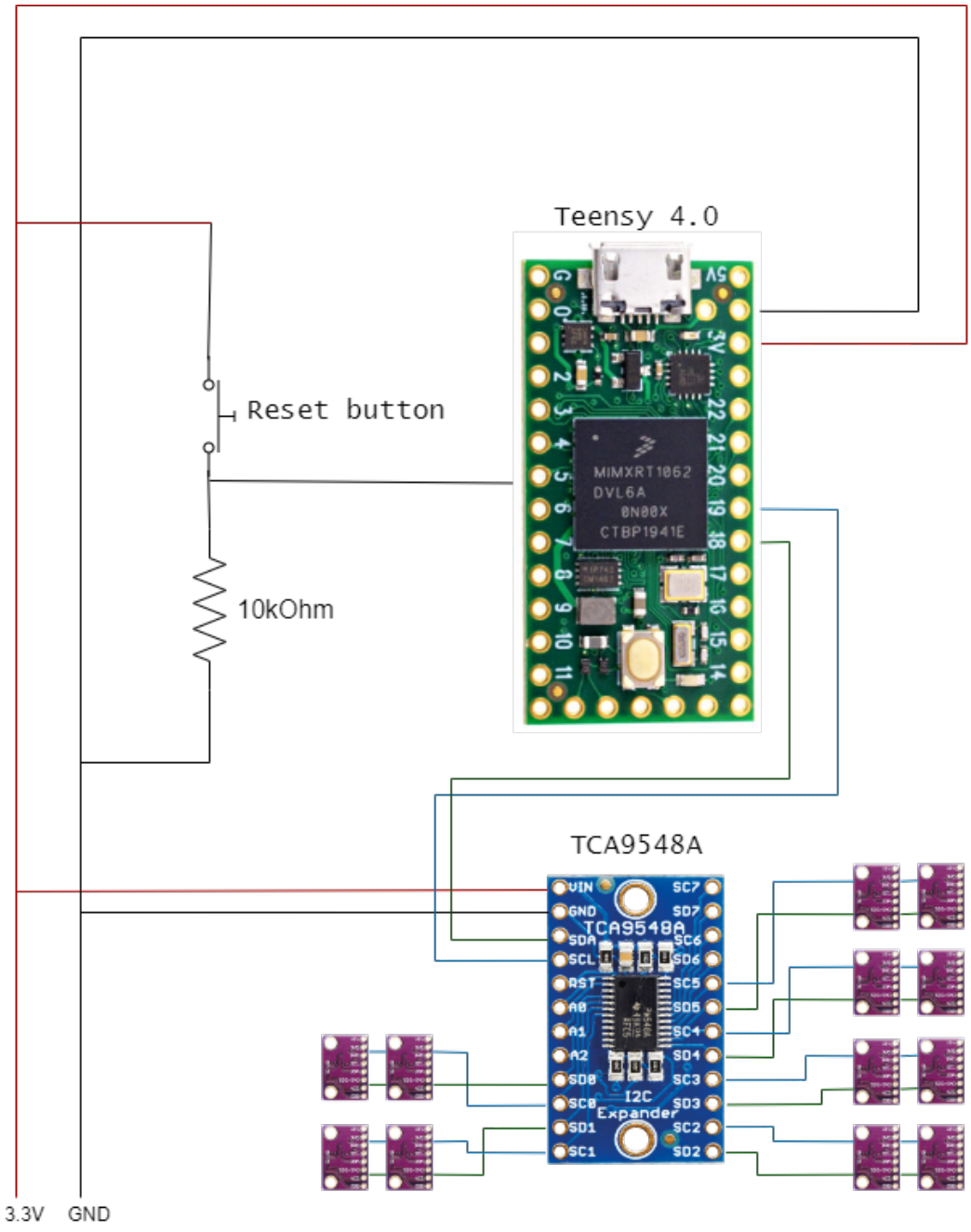

Figure 4.3: Full schematic.

4.2 Software

As mentioned in section 4.1 the program logic runs on the Teensy 4.0 ARM-based microcontroller. One important feature of this microprocessor is the ability to perform a floating point multiplication in a single instruction since the filtering and estimation process is computationally intensive. For this real-time application the reading and processing of sensor data must execute fast for accurate estimations

To develop on this board, PlatformIO was used because it offers a lot of support for libraries, frameworks and a set of tools to get the project running like compiling, uploading to the board and serial terminal tool. If, for any reason, the microcontroller had to be switched for another ARM-based alternative, it would be as simple as changing a board in a configuration. On top of that, it is flexible to choose a framework to work with, in this application the Arduino framework was adopted, it is widely used and tested offering tools like timers, a serial implementation, as well as a I^2C implementation, which are the most used in this application.

Implementing everything from scratch it is an unnecessary challenge and would consume a lot of time. On that account, before starting implementing the code a research was conducted to find the best options, if any, to some challenges. The criteria was simply search open-source projects to integrate and not any sort of external program.

The needed libraries are:

- IMU library to abstract communication
- IMU sensor fusion algorithms library
- Light-weight math library
- Light-weight JSON library

After researching for these libraries the $bolderflight/mpu9250$ MPU9250 IMU library was the first pick, it featured initialization, configuration, reading and all implementations for MPU9250 feature set even though they were not fully utilized. It is well maintained, with active feedback, to answer questions or problems. In addition, this library had a dependency of *Eigen* math library, so to make use of that, the math library of choice was also *Eigen*. The *Eigen* library is popular well-know solution, with a vast amount of essential features but the most needed for this application were the geometry tools, like the quaternions and vector's manipulation. It is mainly stack-allocated, which means it will perform fast on the microcontroller. The JSON library chosen was *ArduinoJson* that is officially in Arduino library repositories. This is a simple library, does not have the cleanest and simplest API but features everything JSON requires and is mainly aimed to maximize performance of embedded platforms. The only library solution not used was one for the sensor fusion algorithms because the solutions found usually came with more complex algorithms like Mahony and Madwick filters and no implementation for a quaternion-based complementary filter, so it was implemented from scratch which it will be discussed later in this section.

4.2.1 Data model

As for the data, it was organized and managed with a Object Oriented Programming (OOP) approach to abstract the logic behind hardware operations and the hand model. Sometimes abstracting hardware implementations is a challenge but Arduino libraries try to solve these issues. In the hardware side the IMU, the button, the timer and the I^2C multiplexer were abstracted in their own C++ classes.

4.2.2 IMU

The IMU class is a wrapper to the integrated *bolderflight/mpu*9250 library responsible for reading but also filter and calibrate. The calibration process consists in sampling the sensor and finding the offsets for the gyroscope and accelerometer with the method mentioned section 3.5.3 and these offsets are stored inside the class. The filtering process is also done inside the reading, where a timer counts the time between readings needed for the gyroscope measurements. The gyroscope readings are somewhat different from usual and are computed and returned as radians instead of radians per second for convenience of the timer living inside the class. The gyroscope delta angles g returned in the reading for each 3 axes is computed as:

```
g = (w - w_offset) * delta_us / 1000000;
```

where $(w - w\_offset)$ is the corrected angular rate and $delta\_us$ is the time between readings in microseconds hence dividing by 1000000 to convert to seconds.

To compute the accelerometer data more steps are needed to account for filtering that is also this class responsibility. In this application, a median filter was chosen, therefore the class needs to store an acceleration buffer of previous readings. The size of the buffer is 12, suggested by quadcopter's implementations, the buffer size determines the low-pass cut-off frequency that was empirically tested to approximately give good results.

After reading the accelerometer data, it is pushed to the acceleration circular buffer followed by the median computation of the buffer returning the filtered acceleration.

```
a_filtered = median(a_buffer);
```

and from the the filtered data the stored calibration offsets are subtracted:

```
a = a_filtered - a_offset;
```

where a is the final acceleration data returned.

4.2.3 I²C multiplexer

The I²C multiplexer class is very simple and boils down to the simple method of changing channels and it depends on the I²C library *Wire*. To avoid unnecessary I²C transmissions, it also stores the current channel so before transmitting it checks if it is already in the target channel.

```
void I2CMux::setChannel(uint8_t channel)
{
   if(current_channel == channel)
      return;

   Wire.beginTransmission(i2c_addr);
   Wire.write(1 << channel);
   Wire.endTransmission();
```

```
    current_channel = channel;
}
```

4.2.4 Hand

For the hand model side of abstraction the class stores 16 quaternions in an array, each one representing the orientation relative to the global frame. This class responsibilities are initialization of joints orientation, update the orientation from the sensor readings and convert orientation from global to last link local frame as discussed in section 3.1, that is only done before serialization.

Each index in the quaternion array matches a joint in the hand, the mapping is as shown:

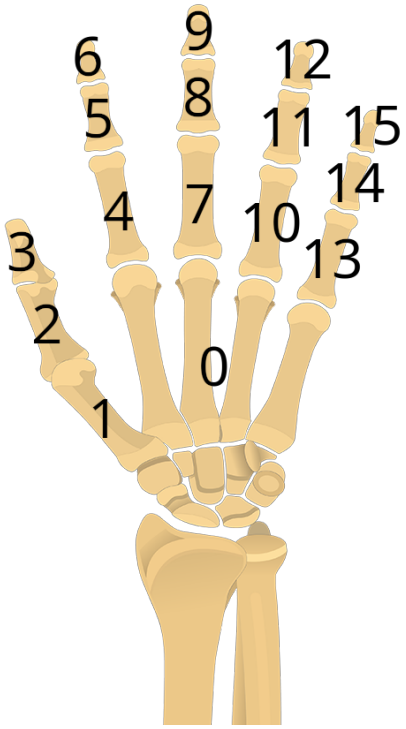

Figure 4.4: Joint indexing (adapted from [48]).

To initialize all joints the angles tilt and pitch are computed from the gravity vector computed from the accelerometer as described in formula 3.5. The gravity vector is the opposite of the acceleration vector of the accelerometer, so it points downwards.

The angles are computed and return the Euler angles vector.

```
Vector3d anglesFromGravity(const Vector3d &gravity)
{
   const double ax = gravity.x();
   const double ay = gravity.y();
   const double az = gravity.z();
   const double ex = atan2(ay, az);
   const double ey = atan2(-1 * ax, sqrt(ay*ay + az*az));
   const double ez = 0;

   return Vector3d(ex, ey, ez);
}
```

Subsequently, the Euler angles are converted to a quaternion and stored in the respective joint using conversion formula 3.6.

The initialization is done only once at setup, but the continuous data acquisition to track orientation is done in the loop block consisting of a reading stage, the update stage and the output stage. The reading is abstracted by the *IMU* class, where it stores the last reading, so it can be fetched later after the reading and all the integration with the *MPU*9250 external library that communicates via I²C. The update stage is the block that calls the model to update given new readings - if the IMU does not have new readings, it skips the update of the respective joint. In the update block the chosen filter is applied, in this case the quaternion-based complementary filter. It inputs the accelerometer 3-axis readings, the gyroscope 3-axis readings and specifies which joint to update.

Firstly the adaptive gain is computed as described in section 3.5.2.

```
void Hand::updateJoint(Quaternion &joint, const Vector3d &dEuler, const
    Vector3d &gravity)
{
   const float em = abs_tp(gravity.norm() - GRAVITY) / GRAVITY;
   float gain_factor;
   if(em <= ERROR_T1)
      gain_factor = 1;
   else if(em >= ERROR_T2)
      gain_factor = 0;
   else
      gain_factor = (ERROR_T2 - em) / ERROR_T1;

   const float ADAPTIVE_GAIN = STATIC_GAIN * gain_factor;

   // complementary filter (...)
}
```

where $ERROR\_T1$ and $ERROR\_T2$ are the relative error thresholds, respectively 0.1 and 0.2. The $STATIC\_GAIN$ is the constant for gain tuned for static conditions and established as 0.005. It is a very low value to respond fast to gyroscope prediction and slowly correct the drift.

To apply the complementary filter, the method in section 3.5 is followed but taking advantage of the math library and changing for some optimizations.

```
void Hand::updateJoint(Quaternion &joint, const Vector3d &dEuler, const
    Vector3d &gravity)
{
   // adaptive gain (...)

   const Quaternion &q = joint;
   const Vector3d &e = dEuler;
   // Attitude propagation
   Quaternion predicted_w;
   predicted_w.w() = q.w() - e.x()/2 * q.x() - e.y()/2 * q.y() - e.z()/2
       * q.z();
   predicted_w.x() = q.x() + e.x()/2 * q.w() - e.y()/2 * q.z() + e.z()/2
       * q.y();
   predicted_w.y() = q.y() + e.x()/2 * q.z() + e.y()/2 * q.w() - e.z()/2
       * q.x();
   predicted_w.z() = q.z() - e.x()/2 * q.y() + e.y()/2 * q.x() + e.z()/2
       * q.w();
   Vector3d predicted_g = predicted_w * gravity;
   // Attitude correction
   Quaternion dqacc = Quaternion::FromTwoVectors(predicted_g, Vector3d(0,
       0, 1));
   dqacc = Quaternion() * (1-ADAPTIVE_GAIN) + dqacc * ADAPTIVE_GAIN;
   dqacc.normalize();

   joint = predicted_w * dqacc;
}
```

The attitude propagation is computed exactly as described in equation 3.4 resulting in the prediction of orientation by the gyroscope (note that gyroscope inputs are already multiplied by the delta time so they are in radians). Then, the gravity vector is rotated to the inverse of the local frame of the gyroscope because the gyroscope prediction is not inverted as the method describes, so to compute the delta acceleration quaternion instead of finding the rotation of "Down" ($Vector3d(0,0,1)$) to the predicted gravity ($predicted\_g$), it is inverted for coherence. After, that the LERP is computed given the adaptive gain and normalized the quaternion right after, because it doesn't maintain the normalized form needed to perform the correction to the predicted orientation of the gyroscope.

4.2.5 Glove sensors

Despite being abstracted within their own class, IMUs still cannot fully describe themselves to communicate, because the information about the I²C multiplexer it is not stored inside *IMU* class. And for the hand, IMUs also do not describe what joint they are attached and tracking orientation. This was solved by having an index map for the I²C multiplexer channels and for the joint index. This solution consists of a basic array that matches the index of the declared *IMU* array with a value, in the case of the I²C multiplexer the channel and for the hand model the indexed joints. Putting the data together the IMUs describe themselves as shown in table 4.1.

IMU	I²C Address	Multiplexer channel	Joint index
0	SECONDARY	0	0
1	PRIMARY	0	1
2	SECONDARY	1	2
3	PRIMARY	1	3
4	SECONDARY	2	4
5	PRIMARY	2	5
6	SECONDARY	3	7
7	PRIMARY	3	8
8	SECONDARY	4	10
9	PRIMARY	4	11
10	SECONDARY	5	13
11	PRIMARY	5	14

Table 4.1

In main program:

```
#define NUMIMUS 12

Imu imus[NUMIMUS] = {
    Imu(&Wire, Imu::I2C_ADDR_SEC),
    Imu(&Wire, Imu::I2C_ADDR_PRIM),
    Imu(&Wire, Imu::I2C_ADDR_SEC),
    Imu(&Wire, Imu::I2C_ADDR_PRIM),
    Imu(&Wire, Imu::I2C_ADDR_SEC),
    Imu(&Wire, Imu::I2C_ADDR_PRIM),
    Imu(&Wire, Imu::I2C_ADDR_SEC),
    Imu(&Wire, Imu::I2C_ADDR_PRIM),
    Imu(&Wire, Imu::I2C_ADDR_SEC),
    Imu(&Wire, Imu::I2C_ADDR_PRIM),
    Imu(&Wire, Imu::I2C_ADDR_SEC),
    Imu(&Wire, Imu::I2C_ADDR_PRIM)
};
```

```
uint8_t mux_map[NUMIMUS] = { 0,0,1,1,2,2,3,3,4,4,5,5 };
uint8_t joint_map[NUMIMUS] = { 0,1,2,3,4,5,7,8,10,11,13,14 };

I2CMux tca9548a(0x70);
Button<HIGH> rstBtn(5); // pin 5, 50ms debounce delay

Hand hand;
```

No layer of abstraction was implemented for this, because the only sensors of the glove are IMUs and can be be easily managed. But, since the data glove has the potencial to feature more data such as touch data or temperature data, for example, it is worth consider abstracting to a *Glove* class where it aggregates the I²C multiplexer and hand model indexes with IMU objects so it can scale better with more and more sensors. Also the serialization is a *Hand* class responsability but as more data is added it doesn't make sense to that data be living inside the class that only tracks orientation.

4.2.6 Program flow

Like most microcontroller's software architecture, this glove program consists of a setup block that only runs once and an infinite loop block that is repeatedly running. Like the flowchart 4.5 describes, the setup block is where the serial hardware gets initialized as well as all the IMUs. Calibration also runs in the setup implementing the process discussed in section 3.5.3 and after that the sensors are ready and the initialization of the hand is computed with the accelerometer data like mentioned in section 3.3.2.

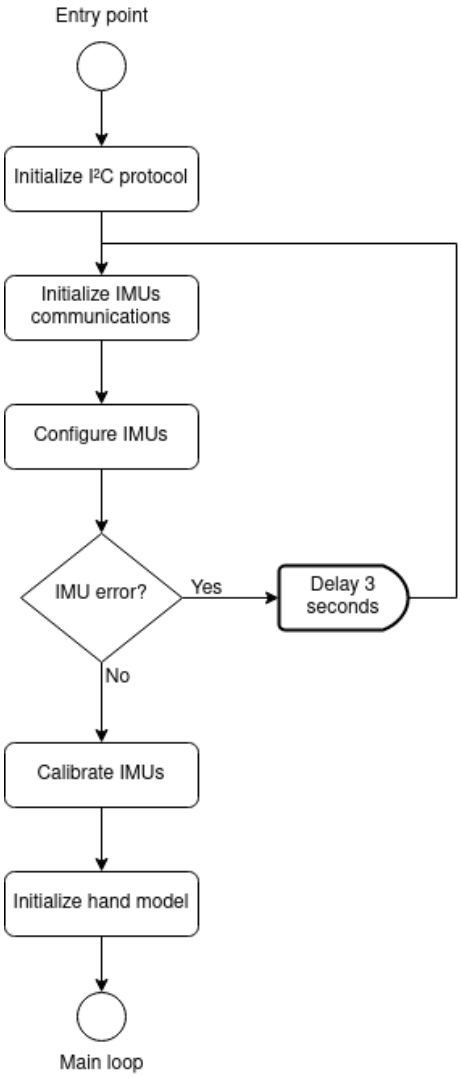

Figure 4.5: Setup flowchart.

After the setup run, the main loop starts and runs at a configurable cycle rate, in fact there is two cycle rates, one for the processing and one for the outputting. This is a solution for the limitations in speed of the serial protocol and serialization method chosen, the JSON format. The JSON format is a plain text format and it is very popular for its intuitive notation and is human readable. That comes with the advantage of being easy to debug, easy to work with and well supported but it comes at a cost of performance which will be discussed later in section 4.3. Consequently, serial protocol struggled to to keep up with processing speeds when more IMUs were added resulting in skipped payloads or malformed payloads. But as a flexible and short-term solution, a cycle rate controller was implemented and limited the outputting of data by an adjustable amount. With all IMUs working simultaneously limiting to 30 frames per second (FPS) or 33333 microseconds of frame time worked very well. This way, the reading and processing could be done at a different rate than the output of data. The reading and processing rate is also adjustable and can be lowered for power saving.

So, the loop consists of reading inputs, including all IMUs and the reset button, after which they are processed applying the estimation computation that updates the hand model orientations and then if the 33333 microsecond time (30 FPS cycle) has passed since last output data was sent, the hand model is serialized and output through serial. The below flowchart 4.6 summarizes the logic:

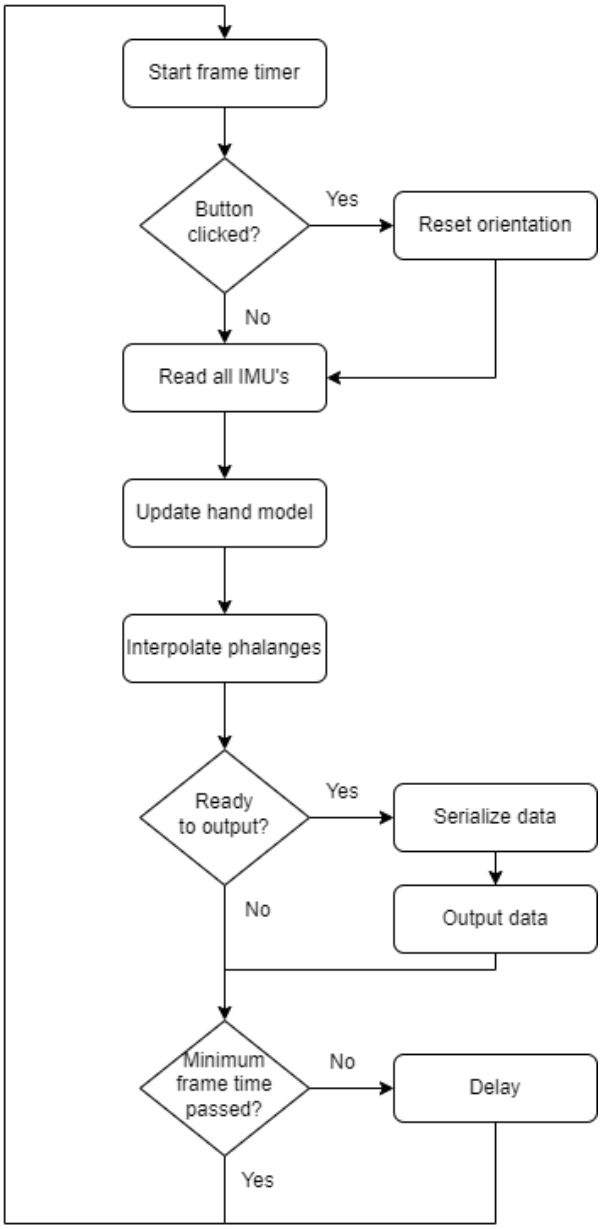

Figure 4.6: Main loop flowchart.

4.3 Glove interface

As mentioned before, the serialization method adopted was the JSON format, a format known for its readability and intuitive structure. Its readability it is due to the only character encoding and the contents can be opened and viewed by any text editor which makes it very simple to debug. The JSON format structure is similiar to *Javascript* object structure, mainly consisting of key-value pairs but with added structure metadata. Because it sends structure metadata such as curly brackets "{}", commas ",", brackets "[]" and '"' plus the key name's characters, the payload length grows a lot with more complex structures and deep nesting, but that is also the advantage of being an intuitive structure, where the nesting is well defined and by reading the key's name one can interpret the payload.

In this application, the advantage of a well defined structure was availed by nesting the information with a semantic meaning. Each orientation is described by a quaternion and a quaternion consists of 4 numbers for "w", "x", "y" and "z" so the object is formed as:

```
{
   "quaternion": {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}
}
```

Note that white-spaces are optional and purely for readability purposes, before sending the payload they are trimmed.

To better understand this application's payload structure let us look at an example of a payload (fully extended open hand):

```
{
   "wrist": {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0},
   "fingers": [
      {"joints": [{"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}]},
      {"joints": [{"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}]},
      {"joints": [{"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}]},
      {"joints": [{"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}]},
```

```
8       {"joints": [{"w": 1.0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1
            .0, "x": 1.0, "y": 1.0, "z": 1.0}, {"w": 1.0, "x": 1.0,
            "y": 1.0, "z": 1.0}]}
9     ]
10 }
```

This way the wrist is semantically separated from the fingers. The "fingers" is an array of 5 elements and each element is a finger indexed from 0 to 4 starting in the thumb (0) and going to the pinky (4). Each finger consists of an array 3 of quaternions that are describing the orientation of the 3 joints in each finger, sequentially, the proximal, intermediate and distal.

Although the JSON format supports scientific notation, the float and double values are extremely lengthy. For example a float in most languages is stored in 4 bytes, knowing that a character is 1 byte, to encode a float in JSON like "1.23456789" takes 10 bytes that is 2.5 times bigger than it could be and grows larger for every decimal place added. In addition to that, the metadata of the structure also adds up to the unnecessary data.

The table below analyzes this wasted space in the payload for variable sizes of encoded floats. Meaningful data is data that is strictly necessary to transmit the orientation. Knowing there are 16 quaternions and each holds 4 floats assuming 4 bytes, multiplying results in 256 bytes of data. The metadata length is fixed and sums up to 436 bytes. The quaternions metadata alone is 21 bytes for each and 336 bytes all combined, the rest is structure of objects, arrays and keys. The tables 4.2 and 4.3 compare the sizes of float representations and how they scale with the payload.

Float encode size	Fixed metadata	Float extra bytes	Float minimum bytes	Total
8 bytes	436	256	256	948
12 bytes	436	516	256	1208
16 bytes	436	768	256	1460

Table 4.2

Float encode size	Fixed metadata (%)	Float extra bytes (%)	Float minimum bytes (%)
8 bytes	46.0	27.0	27.0
12 bytes	36.1	42.7	21.2
16 bytes	29.9	52.6	17.5

Table 4.3

Even for the best case of 8 bytes that corresponds to around 6 decimal places (the dot counts) the useful data only makes up to 28.3% of the payload. The more precision and decimal places are added the more space is wasted in the payload unnecessarily. The payload size in this application is not fixed due to the way the library tries to save bytes encoding and for very small numbers it uses scientific notation in the form of "1.23456789e-8" which adds more characters. After testing,

the average size of the payload ends up around 1200 bytes. So interpolating, each number occupies around 12 bytes which is very significant.

For this reason, the serial port even at 115200 baud rate could not keep up with the reading and processing and would send malformed payloads because it was ordered to send another payload before the last one finished sending. Lowering the maximum output rate to 30 FPS solved the issue but it is unnecessary to limit the output rate for this reason when there iare better options for encoding the data but for fast prototyping JSON was chosen. The JSON format is not the best choice to send streams of data specially numbers, the best scenario is sending just the minimum bytes sequentially and the receiver knows what those bytes represent, however it is harder to debug and *Endianness* could be an issue.

4.3.1 Serial bridge

Ideally, the glove would send data wireless to improve user experience by reducing dangling cables that restrict the movement but for prototyping it is not the easiest solution to start with. So, it was established that the simulator would receive data via User Datagram Protocol (UDP) implemented with sockets. UDP is more appropriate for streams of data because it is fast and simple, unlike TCP that requires handshakes, acknowledgements and error checking. However, the microcontroller had no way to establish connection, so a Python script was implemented to serve as a bridge. The microcontroller is just aware of sending the output through serial and the simulator is just aware of receiving the data through UDP, the script bridges the two.

The serial bridge script is very simple, it just opens the serial port and an UDP socket and then waits to receive data trough serial and sending it through UDP. Fortunally, there are already open source resources that implement both serial and UDP connections.

```
# Create a UDP socket at client side and open Serial port
serverAddressPort = ("127.0.0.1", 5433)
port = "COM3"
with (
    socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM) as
        UDPClientSocket,
    serial.Serial(port, 115200) as serialConn
):
    print("Running...");
    while True:
        data = serialConn.readline()
        UDPClientSocket.sendto(data, serverAddressPort)
```

Chapter 5

Implementation and Results

5.1 Visualization

The simulation and visualization cannot be undermined, it is always crucial for these type of applications. It is possible to debug directly from the output in JSON format and interpret the meaning of the quaternions but it does not scale well nor is a reliable method because it is very prone to error. In the other hand a simulator offers an intuitive interface with humans, easy to interpret for our brains without looking at the numbers being output.

In this application, the support of forward kinematics and graphical context are the main features when looking for a simulation tool. The *Unity* game engine was the pick, it does not feature the most realistic physics and simulations, because it aims to be fast and compromises some accuracy simulating, but for the needs of the application it does not need to actually simulate a lot of the real world, it just needs to set orientations, locally and globally, of the objects and output somewhat realistically to the screen. *Unity* is a very popular game engine because it is free and one of the simplest to learn but still full of features. The documentation is very mature and the scripting language *C#* it is not limiting in any way for this application. The language *C#* comes with an UDP implementation, which is also a bonus.

A working simulation was the main goal during the first stages of development, as stated before, the rest of the development progresses faster with a visualization tool. Firstly, the features were tested to reaffirm that the simulation fits the application needs. An empty 3D project is created and only has a scene with a directional light and a camera, so the first step is to create the hand model and it was done from scratch using *Unity*'s native 3D objects. To control the position, orientation or scale of the objects in *Unity* the transform component is used. This component has a tree structure, where it holds children's transform and it is how it computes forward kinematics under the hood. When the parent transform moves, all its children are attached and also move.

To construct a hand, the transform and object hierarchy is rooted in the arm, followed by the wrist, the palm is attached to the wrist and the base of the fingers are children. Each finger then has one child, the proximal and each proximal the intermediate and the intermediate holds the distal. The tree structure is better visualized in figure 5.1.

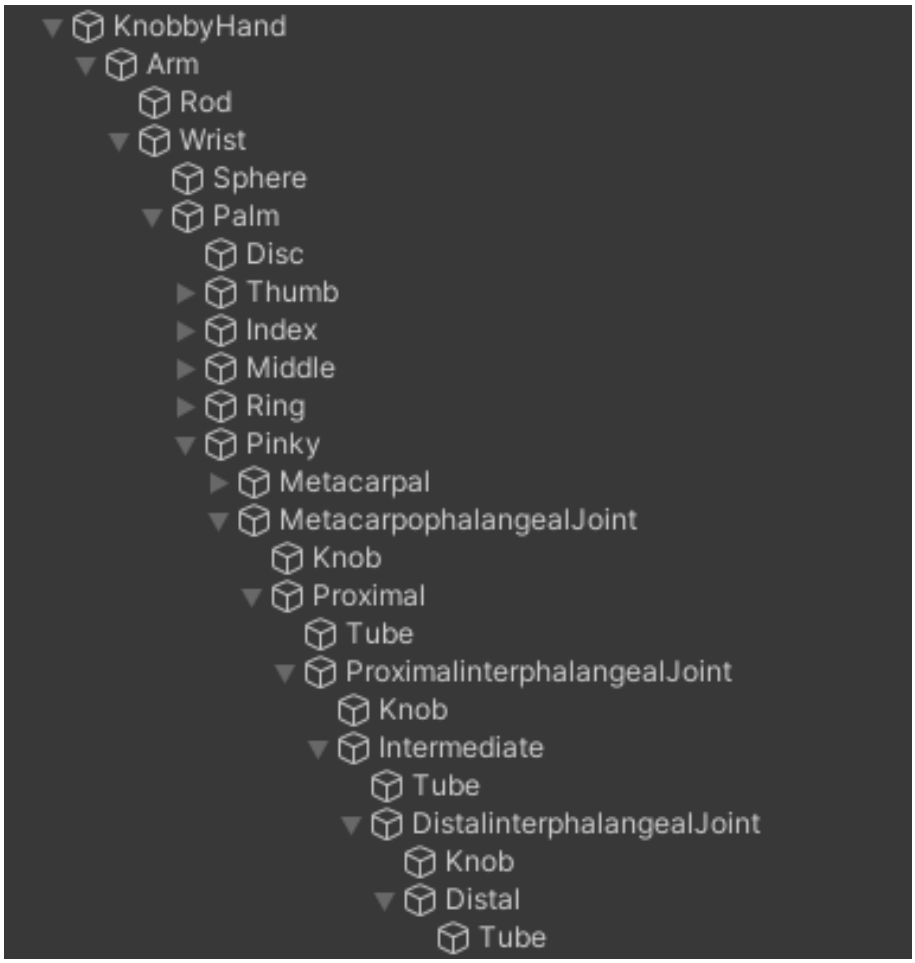

Figure 5.1: Transform hierarchy model.

The rods, spheres and disc objects are attached because they hold a mesh renderer component that is what it makes it possible to visualize, the objects that the transform will be manipulated just hold the transform. For default, these *Unity*'s prefabs come with capsule collider and needs to be removed or disabled. The end result is shown in figure 5.2.

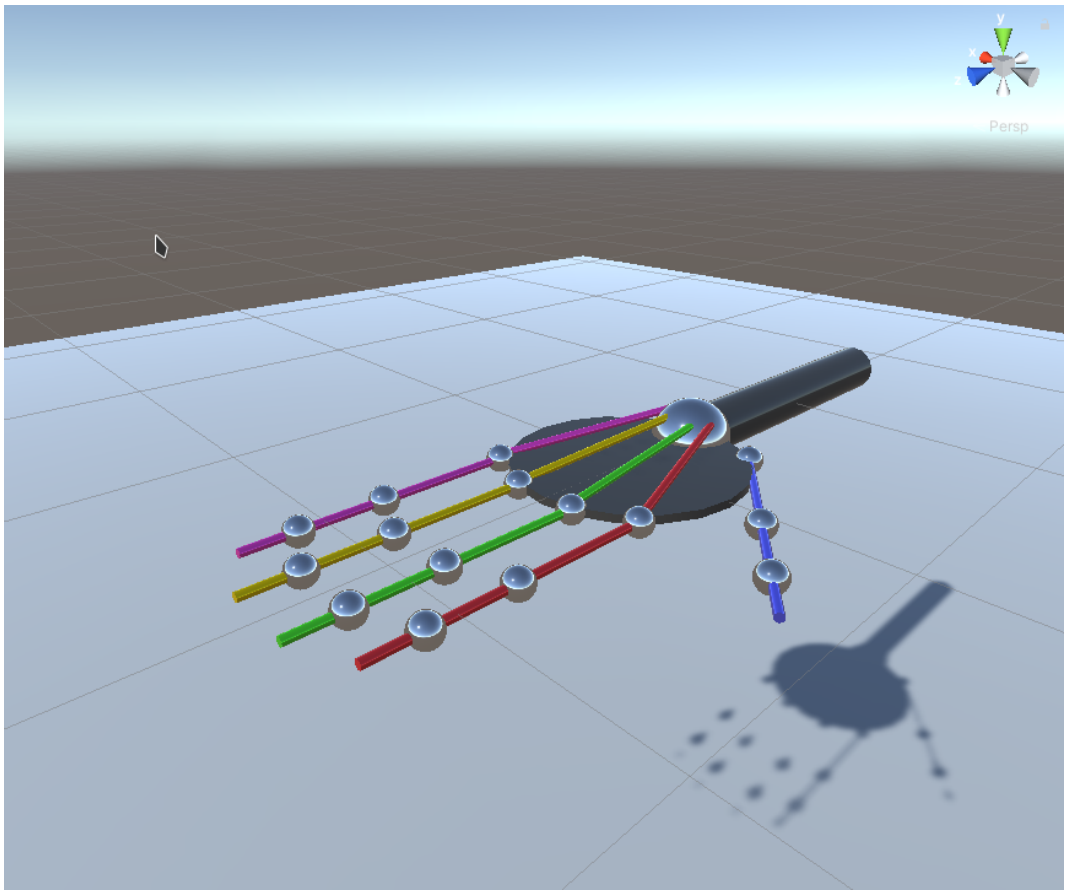

Figure 5.2: Visual representation of the hand.

To test it, the hand controller script was created and attached to the hand object, this way it is possible to manipulate the transform and its children. The test consists of just rotating the wrist and folding the fingers on loop at a given velocity. The test was successful and proved that is capable of setting the orientation relative to the parent that is the same format as the output of the microcontroller.

The scripting of the *Unity* game engine is very complete and supports almost every feature of the *C#* environment including UDP library. Because the game engine calls the function "Update" in the script component every frame, if the call to read data from the UDP socket is blocking then it blocks the loop until the data arrives which is undesirable. Ideally the simulation runs continuously and when new data arrives, it updates and when there is no new data it can perform other tasks. Usually reading streams from sockets blocks the program but the library also supports asynchronous programming which fits well with the nature of the simulation logic. It is an event based system where it is defined the event to listen for and a handler (usually a function) to execute when the event is triggered. In this application, the event to listen for is received data in the UDP socket. When the data is received, it is stored in a variable accessible by the "Update" method of the script, usually inside the class's script. When it is received it also flags it to let the "Update" method know it needs to parse that data and update the hand object. Because it runs

asynchronously, race conditions must be handled with care, if the data is received while parsing the previous data it is possible to corrupt the data in the middle of parsing. So mutexes were used when writing or reading the shared resource assuring atomicity of the operations, blocking access of other operations to the shared resource while another operation is handling it. This asynchronous approach, summarized in figure 5.3, makes it possible to decouple the speed of the microcontroller output and the game engine speed, solving the issues of synchronization.

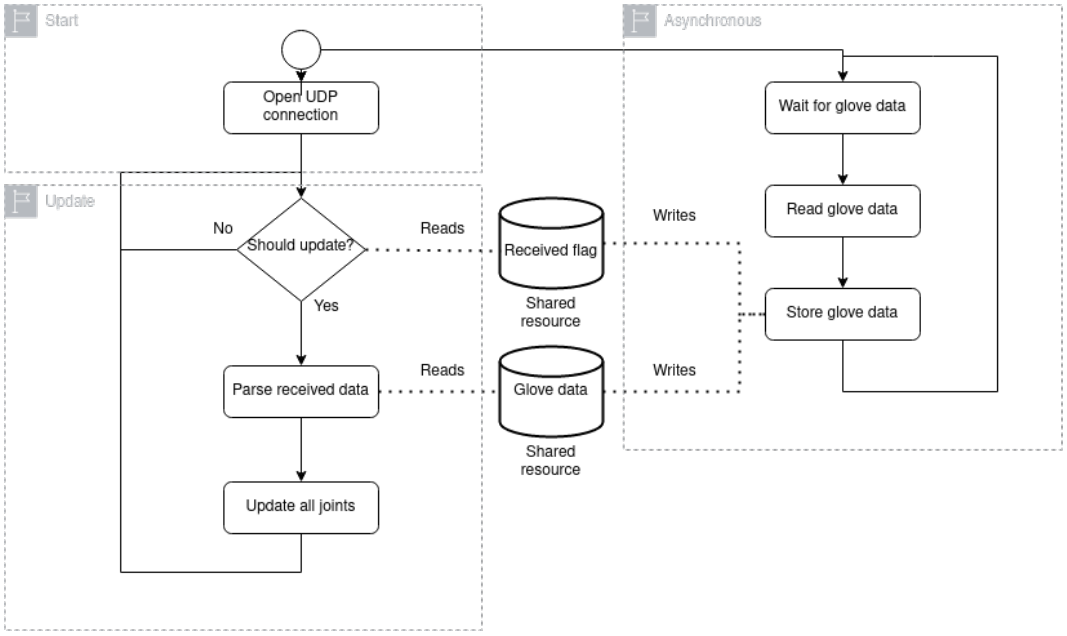

Figure 5.3: Simulation program flowchart.

To test the UDP connection, a *Python* script was implemented, that inputs a user string and sends it through UDP to localhost. The server side (*Unity* script) reads it and logs the result. The test revealed it was a viable solution, also capable of reading continuously streams of data.

```
while True:
    msg = input("")
    bytesToSend = str.encode(f"- {msg}")
    UDPClientSocket.sendto(bytesToSend, serverAddressPort)
```

It was also tested if the solution could handle non-stop streams of data simulating high frequency data outputting. So, a script that just sends the value of a incremental counter in an infinite loop was implemented and the asynchronous solution, in fact, was working properly.

```
counter = 0
while True:
    bytesToSend = str.encode(f"Message: {counter}")
    UDPClientSocket.sendto(bytesToSend, serverAddressPort)
    counter += 1
```

The two last tasks the script must perform are the parsing of the received data and the update of the transform of the correct objects. The *Unity* game engine has its own serialization implementation and includes JSON serialization and deserialization tools. It is only required to define the structure in a *C#* class or struct, add the "Serializable" *C#* attribute and the JSON tools create the object from the JSON encoded string.

```
[Serializable]
public struct FingerPose
{
    public SQuaternion[] joints;
}

[Serializable]
public struct HandPose
{
    public SQuaternion wrist;
    public FingerPose[] fingers;
}
```

The "SQuaternion" type is a custom serializable quaternion because the *Unity* quaternion type does not support serialization by default. It just stores w, x, y and z values and implements implicit operators to allow an easy conversion between the native quaternion type supported by *Unity* so it can be used like so:

```
SQuaternion s_quaternion = new SQuaternion(0, 0, 0, 1);
Quaternion quaternion = (Quaternion)s_quaternion;
```

To perform the rotations on the objects, *Unity* handles it when setting the property "localRotation" on the transform component. It expects a quaternion and sets its orientation in relation to the parent transform. One issue that immediately surged is that the microcontroller and the simulation were not using the same reference frame. The figure **??** helps visualize the frames of reference.

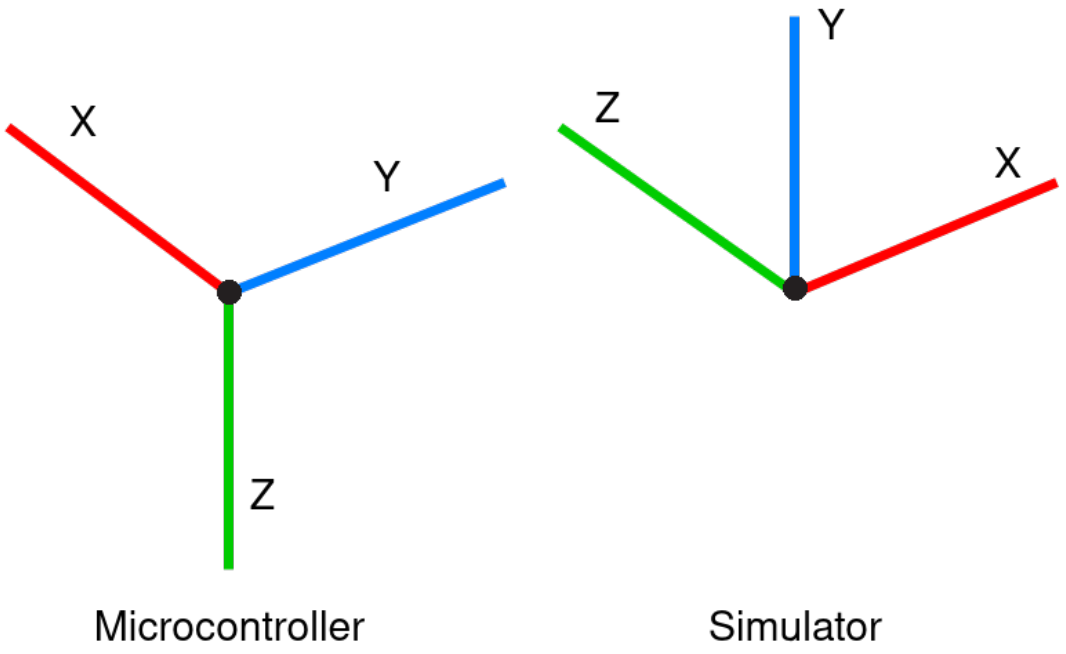

Figure 5.4: Microcontroller and simulator frames of reference.

The frame of reference transformation is defined by the transformation matrix from microcontroller frame to simulator frame ${}^{S}_{M}\mathbf{T}$:

$$
\begin{aligned}
{}^{S}_{G}\mathbf{q} &= {}^{S}_{M}\mathbf{T}\,{}^{M}_{G}\mathbf{q} \\
\begin{bmatrix} w_s \\ x_s \\ y_s \\ z_s \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} w_m \\ x_m \\ y_m \\ z_m \end{bmatrix} \\
\begin{bmatrix} w_s \\ x_s \\ y_s \\ z_s \end{bmatrix} &= \begin{bmatrix} w_m \\ y_m \\ -z_m \\ x_m \end{bmatrix}
\end{aligned}
\tag{5.1}
$$

So, before setting the local rotation of the objects, the frame of reference transformation is computed inside the function "ToFrameOfReference". It takes the microcontroller quaternion as an argument and returns a new quaternion in the simulator frame of reference:

```
private Quaternion ToFrameOfReference(Quaternion q)
{
    return new Quaternion(q.y, -q.z, q.x, q.w);
}
```

And finally assembling the parsing and the rotation of the objects, each frame, if it has new data, runs:

```
HandPose pose = ParseGloveData();

wrist.localRotation = ToFrameOfReference((Quaternion)pose.wrist);
for (int i=0; i < 5; i++)
   for (int j=0; j < 3; j++)
      fingers[i].joints[j].localRotation =
         ToFrameOfReference((Quaternion)pose.fingers[i].joints[j]);
```

To test that everything was properly working, a test very similiar to the previous one was performed but sending valid JSON data. It also takes input from the user to the target angle of the wrist and form the JSON payload with all other quaternions as identity quaternions, which means no rotation. The test successfully validated the correct parsing of JSON data and the correct assignment of the orientation in the right frame of reference.

5.2 Real Prototype

The glove design presented a challenge in two fronts, glove choice and wiring solution. Ensuring the sensors are tightly coupled with their respective links to accurately capture the movement boils down to the glove choice - if the glove is loose on the hand the sensors attached do not accurately follow the movement but stiffness helps keep the sensors steady. Gardening gloves were the first idea to try, but it was decided they were often too loose and certain gestures like lifting the proximal would create a fold exactly below the sensor. In the other end of the spectrum surgery-like gloves that are very tight and were considered but mounting the sensors is not straight forward. So a glove made out of wool, shown in the figure 5.5 was tried because it gave the most options, it was tighter than the gardening ones and the sensors could be stitched to it. The sensor location is flexible to some extent in contrast to the others that could only fit one way.

Figure 5.5: Glove in the hand without sensors.

I²C communication protocol only requires 2 connection and is the best case scenario wire-wise but the sensors also need 3.3V power and ground, and half of them need to pull the I²C address pin high. In total, half the sensors need 4 wires and the other half 5 wires for the total 12 sensors. This amount of sensors in a such a small space results in the wires bumping each other, like shown in figure 5.6, and inadvertently moving the sensors causing false estimations. This solution is not optimal and to get around the problem a bus-like wiring was attempted. Because sensors are all powered by the same device and voltage, and the I²C works as a bus, it allows for a cascading wire layout where the intermediate sensor connects to the proximal and proximal to the multiplexer reducing the floating wires roughly by half, like shown in figure 5.7. This also reduces the chance of long wire communication instabilities although it did not reveal being an issue. This design has a flaw of overloading the current of the power in the wires nearest to the source because the sum of the currents drawn by sensors flows there. Increasing the current the voltage drop in the wire increases proportionally and the voltage at the further sensors gets lower and the fluctuations can cause instabilities. In this design that did not reveal being an issue, because only two sensors get cascaded, except the thumb that cascades 3 sensors.

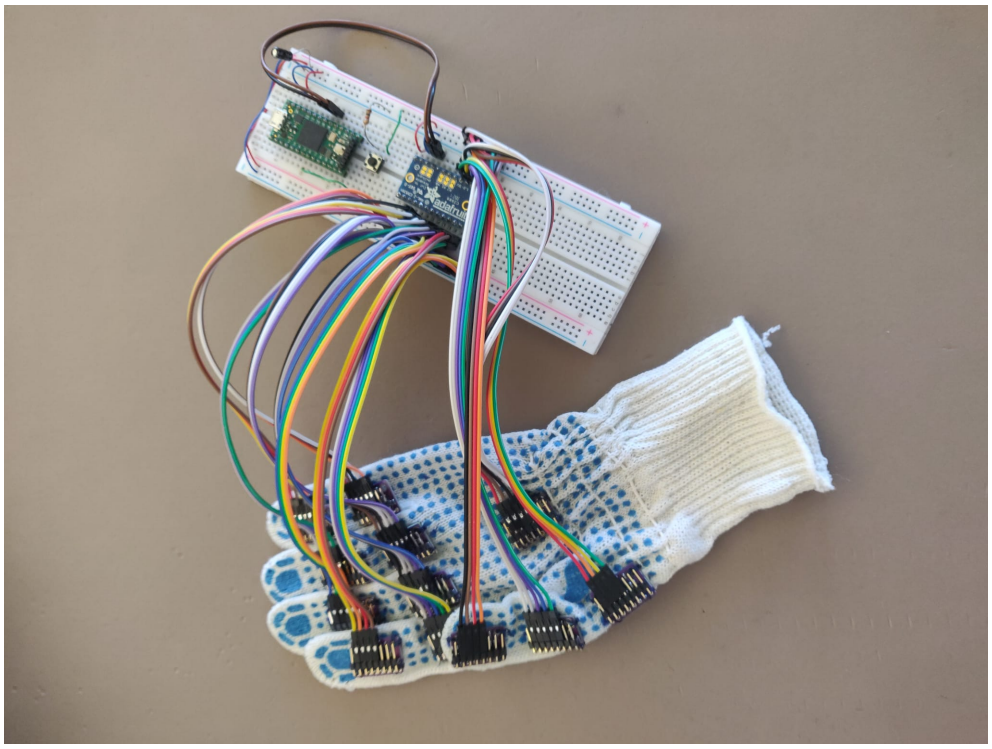

Figure 5.6: First glove design with individual wiring.

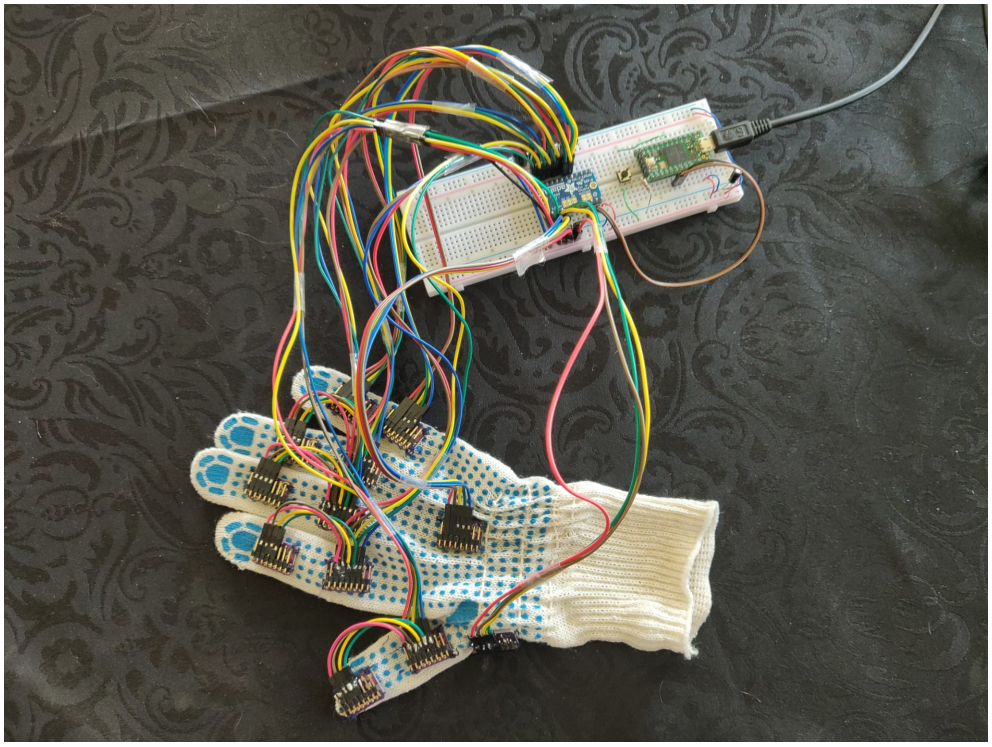

Figure 5.7: Second glove design with cascading wiring.

5.3 Offline calibration

The offline calibration was tested for multiple sample sizes namely 10000, 1000 and 400. All calibrations setups consisted in leaving the sensors in stable positions during the sampling. The offline calibration program simply reads a sensor at a time the number of samples and then sends in a CSV format through serial to the computer where the offline calibration script is running and reads the serial data and dumps it to a CSV file. That file can later be analyzed, for example by *Excel*, to compute the mean of the samples.

Instead of sampling separately for each sample size, the sensors were sampled just for one size, 10000 samples, and the mean was computed for 400 first samples and 1000 first samples to mitigate the effect of temperature variation between the 3 samples. The 10000 sample served as ground truth and the relative error of the other 2 samples was computed to analyze the effect in sample size.

Although the temperature was also sampled, it was not profiled like suggested in [37] so, the results of the offline calibration were poor, the offset estimation is inconsistent, working well in the same day and poorly in the next day, proving that there is no fixed offset for the gyroscopes, which leads to the conclusion that environmental factors like temperature and pressure play a role in the gyroscope offsets [37]. So, the online calibration was adopted and showed better results.

From the tables 5.1, 5.2 and 5.3 it is better understood the flaws of this calibration method. Apart from the temperature factor, the samples reveal inconsistency when estimating the zero offset of the gyroscope, mainly due to environmental factors and the user. A small unintended movement can cause a significant error, in this experiment up to 40.42% between the 400 and the 10000 sample. Overall, the more samples the less an error impacts the estimation but the chance of a unintended movement increases and the fact that 10000 samples takes around 5 minutes to collect is not reliable. The balance between the time it takes to calibrate and the performance of the calibration is better tested empirically than analytically by testing the hand orientation estimation performance. Below 1000 and above 200 samples seemed a good range to test because above 1000 samples the time to calibrate heavily punishes usability and below 200 samples the drift is somewhat noticeable to the naked eye. For these reasons, this method of calibration, both offline and online, is not the best, but for quick prototypes is very suitable for its simplicity. On the fly online calibrations aim to solve these issues by estimating the offsets during usage.

IMU	X offset	Y offset	Z offset
0	-0.0057850070	0.0206230885	0.0019162270
1	0.0131991482	-0.0171678512	0.0098898533
2	-0.0007668956	-0.0061865114	-0.0070752751
3	-0.0174087112	-0.0332459188	0.0068540163
4	0.0065506244	-0.0492009477	0.0064423920
5	0.0105264648	0.0041178427	0.0004415588
6	-0.0131047644	-0.0566154085	0.0011297302
7	-0.0173410659	0.0564493313	0.0170428938
8	-0.0124964891	0.0165936646	-0.0118663752
9	-0.0170175400	-0.0088691015	0.0056922211
10	-0.0102417155	-0.0057028739	0.0006573847
11	-0.0045634495	0.0415907944	-0.0064489966

Table 5.1: 10000 sample results.

IMU	X relative error	Y relative error	Z relative error
0	6,79%	0,35%	9,91%
1	0,23%	0,21%	1,99%
2	32,97%	0,36%	0,81%
3	0,31%	0,30%	0,24%
4	0,37%	0,28%	0,02%
5	0,97%	2,80%	14,92%
6	1,57%	0,42%	2,49%
7	0,35%	0,10%	0,06%
8	0,67%	0,52%	0,86%
9	0,44%	0,34%	1,46%
10	3,30%	1,74%	27,47%
11	1,05%	0,28%	0,50%

Table 5.2: 1000 sample relative error.

IMU	X relative error	Y relative error	Z relative error
0	7,82%	0,57%	15,34%
1	0,41%	0,18%	1,86%
2	40,42%	0,14%	1,11%
3	0,44%	0,29%	0,08%
4	1,67%	0,34%	0,12%
5	1,78%	2,52%	15,17%
6	2,19%	0,57%	6,29%
7	0,58%	0,17%	0,45%
8	1,23%	1,00%	0,65%
9	0,43%	0,82%	0,71%
10	2,86%	0,73%	25,64%
11	1,12%	0,36%	0,47%

Table 5.3: 400 sample relative error.

5.4 Glove usage

The first step to use the glove is making sure the microcontroller has the software uploaded, this is required only once. After connecting the microcontroller to a computer via USB, the *Python* script "serial_bridge.py" can be run passing the USB port to which is connected as a command line argument, like shown in figure 5.8. On *Windows* is commonly "COMx" and *Linux* can be for e.g. "/dev/ttyACMx" or "/dev/ttyUSBx" where "x" is some number. This script will bridge the serial communication to UDP and serve as a debug console as well. If the serial communication stops or there's an error the script needs to be rerun. Note that the script requires *Python* version 3.10 or upwards installed.

Figure 5.8: Example to running "serial_bridge.py".

After starting the script the user needs to stand still during initialization and calibration of the sensors. This information and the stage of the program is printed in the terminal and also logged in the simulator console.

Afterwards, to visualize, like in figure 5.9, the Unity project "HandSim" is opened and press the play button in the editor to start the simulation. The simulation can be started at any point, even when the glove has not started and can be restarted at any point. This is one advantage of the asynchronous implementation and the UDP protocol that does not need connection handshakes.

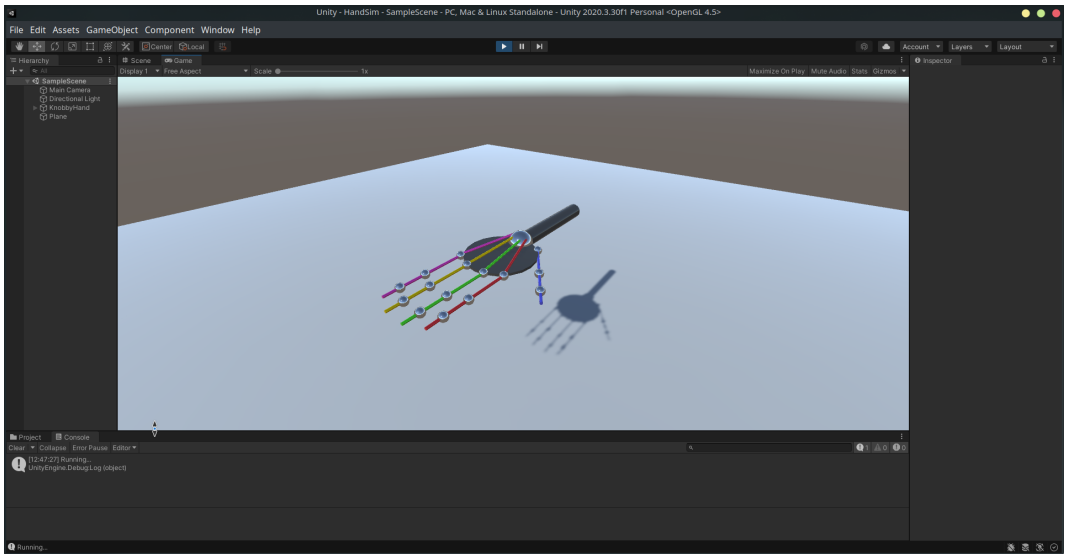

Figure 5.9: Example to running *Unity* simulation.

To reset the pose of the model the button can be pressed, this is useful mainly to correct the drift in the heading without the need of a restart. This uses the accelerometer in the same way the hand is initialized the first time.

On a final note to the usage, it is important to try to accommodate the glove in a way that the IMUs lay on top of the phalanges. The model of the hand has that in consideration and this way the orientation is more accurately tracked.

5.5 Hand orientation estimation

The results of the orientation estimation solution were evaluated empirically by comparing the movement of the user's hand with the simulation. A set of commonly used hand poses were performed to test the capabilities and limitations of the current solution.

In figure 5.11 the pose of the hand is accurately depicted. This pose is useful to test basic functionalities because it does not feature complex orientations, two fingers are fully curled and other two fully extended, and the thumb to the side near to the initial position.

In the second demonstration (figure 5.13), where the index and the thumb are connected, presents other challenges and reveals some imperfections. In figure 5.13, the fingers do not connect and does not represent the orientation as accurately. This is mainly due to two issues: the wires and the model. The wires are sufficiently stiff and fabric does not keep its shape, then allowing it to be pulled so the sensors are not in the same orientation as the hand causing the error. This error is common when curling the fingers and also when the wires of another sensor obstruct other fingers movement and stretch the fabric. The figure 5.15 also confirms it, as the fingers should be slightly more curled. The other issue is the thumb in the model, it is very far and the scale is not adjusted because the model used for all the fingers is the same. In the other four fingers it is not very noticeable but in the thumb the simulation is not represented so well, however the orientation

matches. The last demonstration 5.17 combines some poses like curled, extended and connecting fingers, and the straight fingers (index and pinky) show the ability to track lateral movement. This pose also shows the wire problem, where the pinky finger is obstructed by the wires of the ring finger, in this case the orientation estimation was not affected but it is important to note the issues in user experience.

The demonstrations are fallible because of the repeatability in initial position and calibration, and because the assumptions of the position a anatomy in the simulation environment. This can be easily solved by adjusting the finger scale and placement mainly the thumb. However the results were still satisfying proving the concept and revealing the main issues to address in future work.

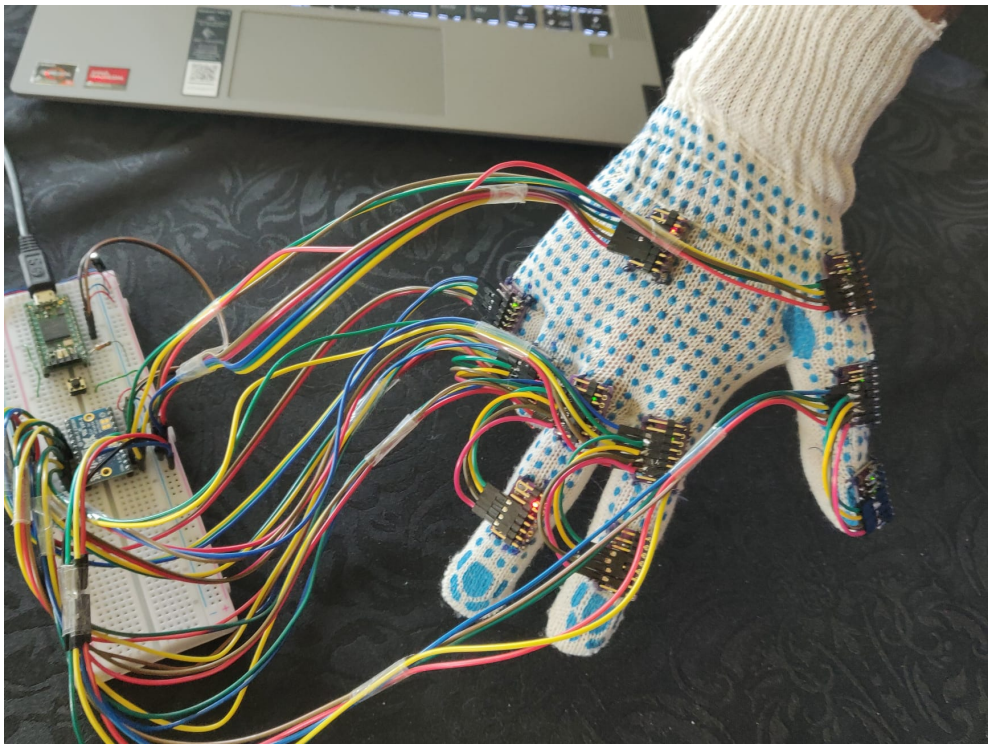

Figure 5.10: Two fingers demonstration.

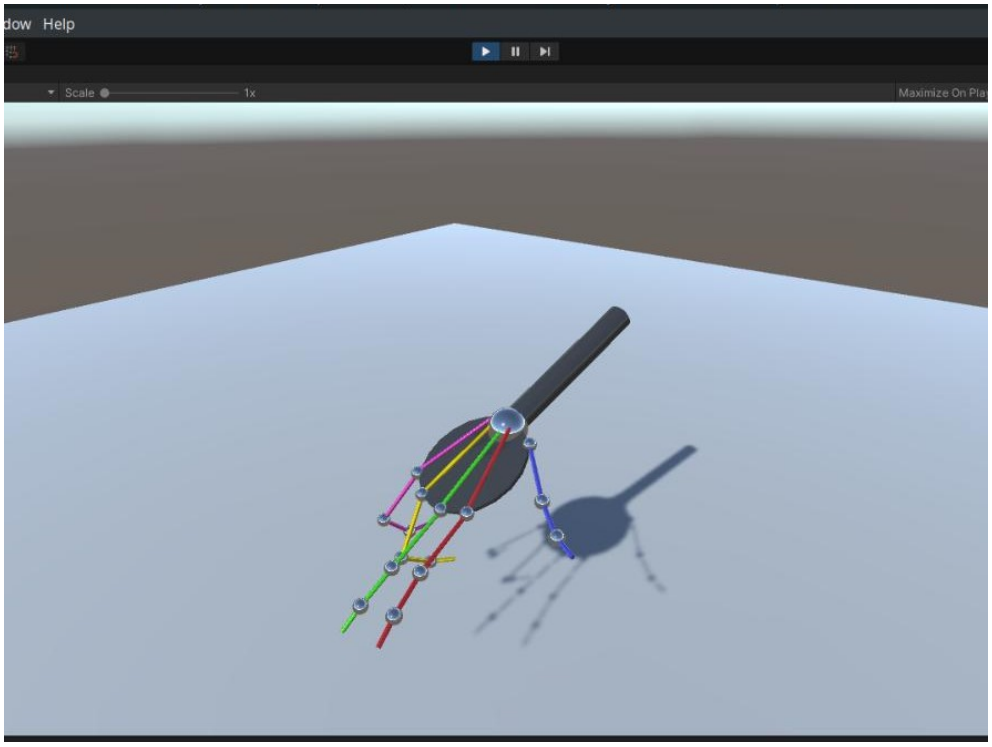

Figure 5.11: Two fingers simulation.

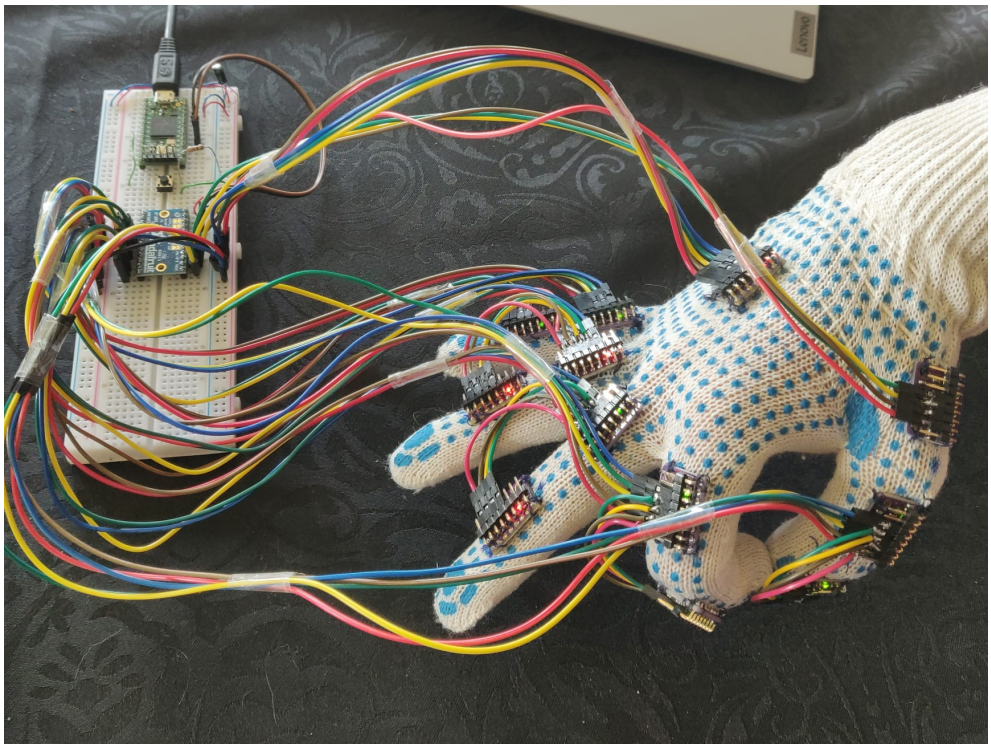

Figure 5.12: Index and thumb connected demonstration.

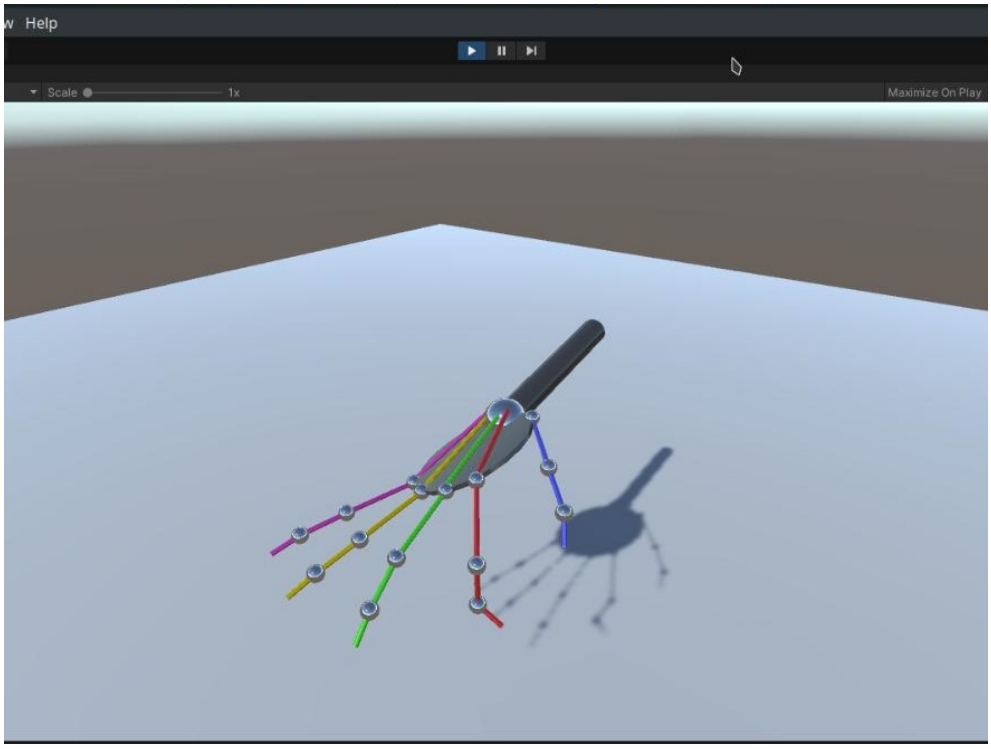

Figure 5.13: Index and thumb connected simulation.

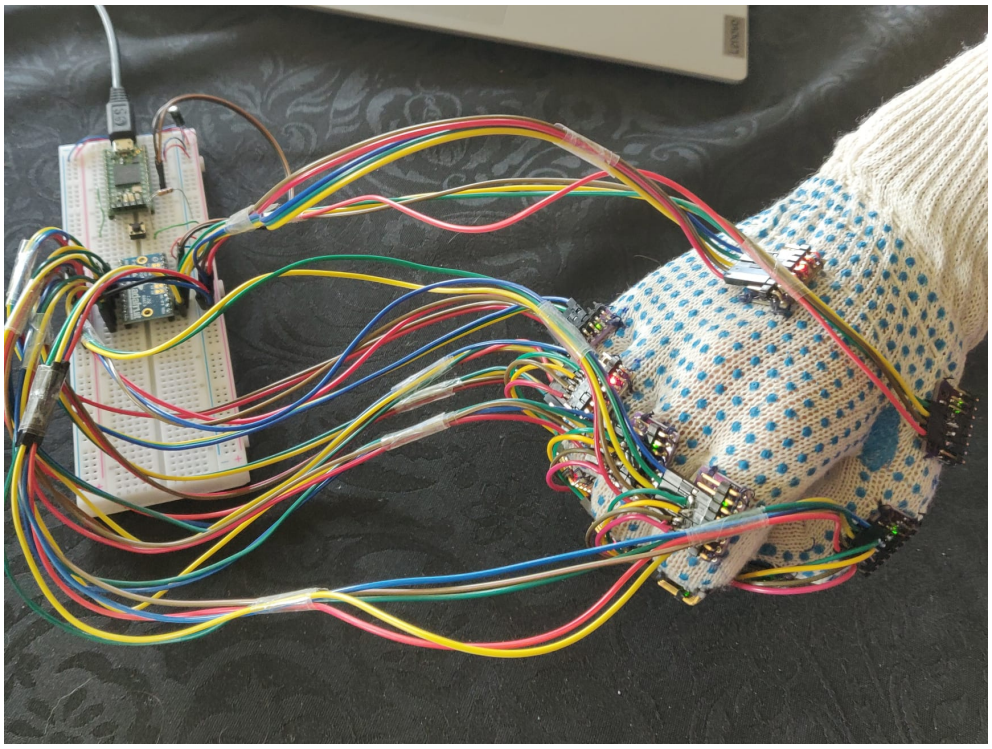

Figure 5.14: Closed fist demonstration.

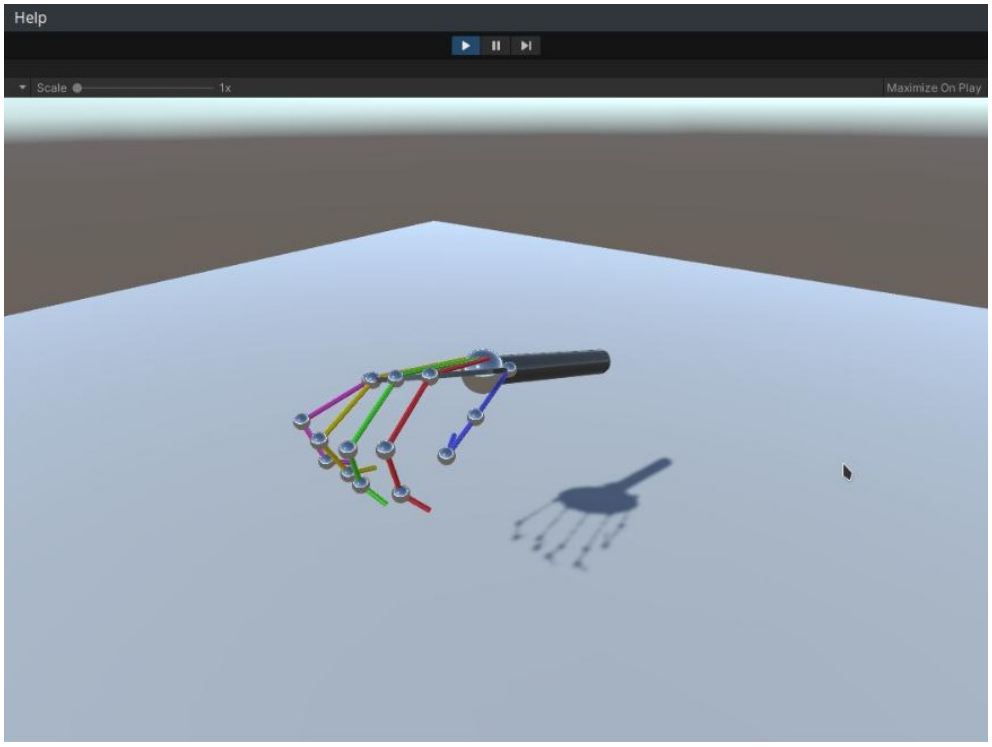

Figure 5.15: Closed fist simulation.

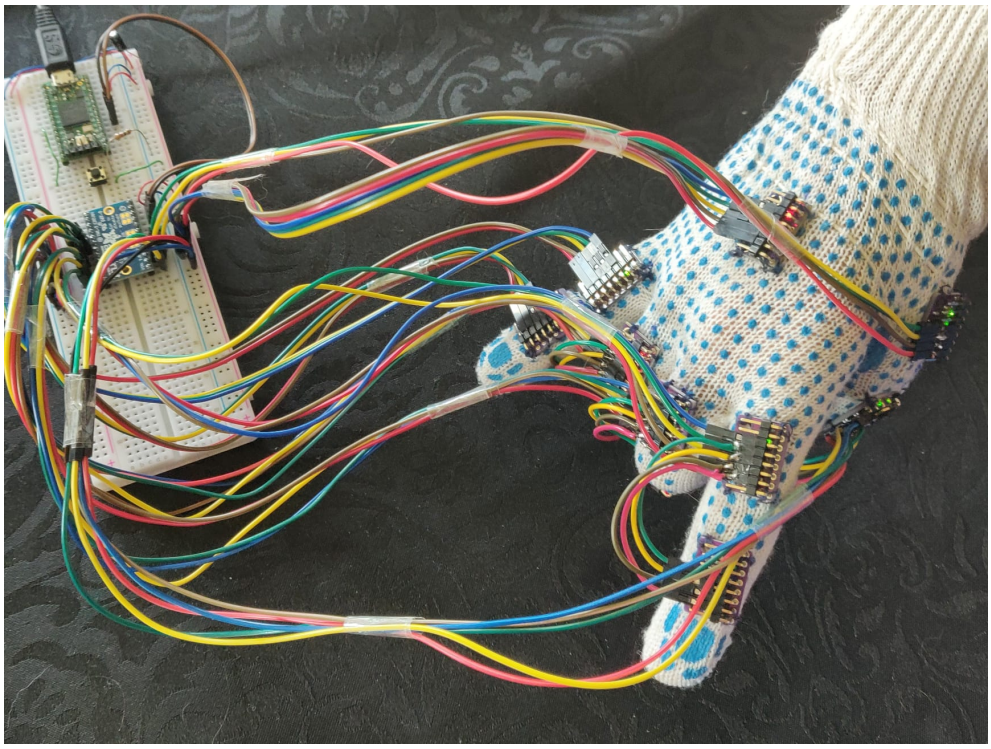

Figure 5.16: "Spider-man" hand demonstration.

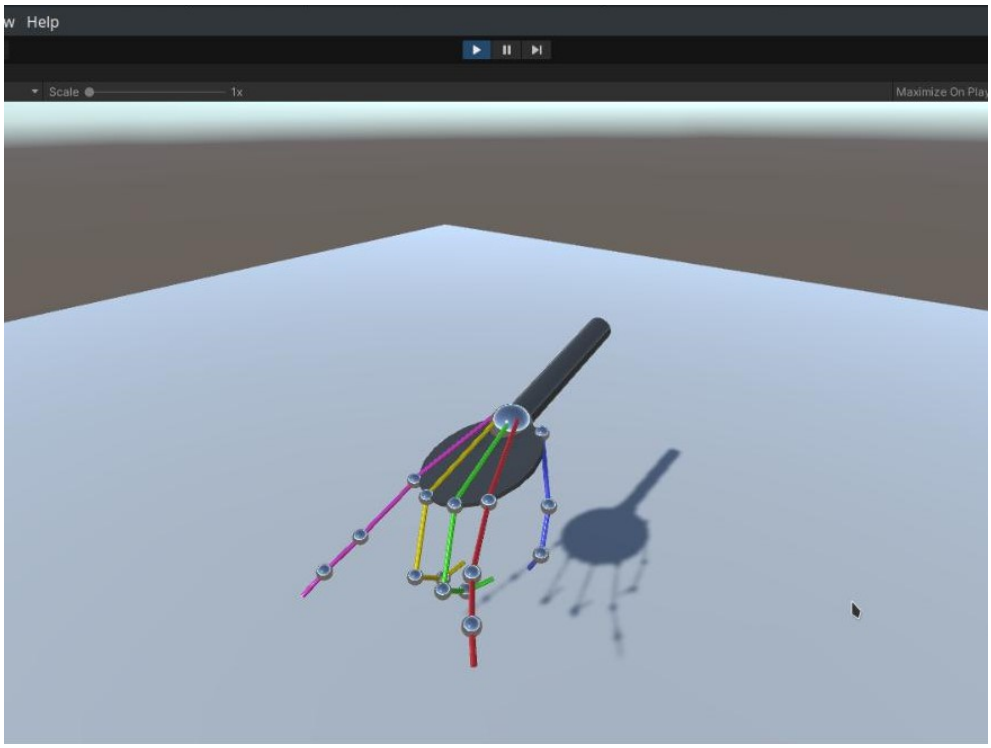

Figure 5.17: "Spider-man" hand simulation.

Chapter 6

Conclusion and Future Work

The current glove design is not the most flexible in terms of sizes and reliability. Although the the fabric was chosen to be flexible it is also a problem, the same way it facilitates adjustments, it also enables errors in positioning. The implementation of the orientation estimation is error tolerant to a point but assumes the IMUs are on top of the phalanx nearly flat. The fact that there is only one glove size presents a problem for smaller and larger hands. A possible solution is that, instead of attaching the sensors to the glove make it modular in a way the sensor array can be worn on top and adjusted for any kind of glove or even without glove. Some mechanism like strapping it around the finger might be viable but the palm sensor may need another solution.

As a prototype, the simplest communication protocol was serial but it is wired, which sometimes may limit flexibility or restrict the user's movement and, generally, wireless (if inside performance requirements) is preferable. The two main options of wireless solutions are changing the microcontroller for one that features a wireless solution or adding a module for the microcontroller to interface. A popular module is the *ESP*8266 which can be interfaced by SPI or UART and it is fairly simple with a lot of documentation. Adapting the current solution for any of the two proposed wireless solutions is straight forward. If the microcontroller were to be swapped, the only requirement is to keep the ARM processor architecture not to cause conflicts with libraries but *PlatformIO* would handle the rest. In both solutions, adapting the output function is the main change to interface with the chosen wireless feature. Ideally, the wireless solution is WiFi to connect to the simulator network and communicate directly with through UDP but it is possible adopt another wireless solution like *Bluetooth* and implement a *Bluetooth* bridge like the one implemented in *Python* for serial.

Binary encodings are the most efficient sending data and are very popular for real-time applications that are robust to errors, meaning if there is an error in transmission, the system is not fully compromised or there are mechanisms computationally cheap to detect errors and ignore those payloads. In this application, the priority of the glove is to stream the data fast and it is assumed that the responsibility lies of the receiver to handle errors. For these reasons, binary encondings

fit the application's requirements and serialization should be optimized from the current JSON enconding. A promising solution to better serialize is *Google*'s *ProtocolBuffers*, it is a tool for serializing cross-platform and cross-languages and compiling it produces the code for the specified languages, meaning there is no metadata being sent and both ends of the communication know the structure offline. The idea was not explored in this application but revealed being very suitable to solve the JSON serialization problems.

The distal phalanx interpolation was not vastly explored leaving room for improvement, considring that the linear interpolation implemented in this application was simple. Possibly with more anatomical research, a better estimation can be achieved. There is also the possibility of not interpolating it but tracking the distal phalanx for each finger by adding 4 more IMUs (tha same way that the thumb's distal phalanx is being tracked). The I²C multiplexer still has 2 empty channels that can connect up to 4 IMUs so there is no need to expand.

To solve the heading drift the most common solution is by adding a magnetometer and add a heading correction step in the filter. However, in this application, the glove was mainly targeting industrial application and in these environments magnetometers struggle to accurately measure heading due to high soft and hard iron disturbances, even with calibration, so it was ruled out from the beginning. Other solutions were not explored but the current implementation leaves room for the heading correction without remaking any of the current work. In fact, only proximal phalanges have that degree of freedom and it is possible to simplify the problem even further by reducing the degrees of freedom of other phalanges that can only move up and down but currently all movement is tracked. A possible approach is looking at sensors that measure angles in just one axis like flex sensors. They are not very precise but, since they are just required for a slow correction, it is not a problem. However, their placement in a glove can be challenging, because there is no easy pivot point laterally.

Calibration revealed being a big role in IMUs precision and accuracy and even more relevant to maximize performance of low-cost sensors. As mentioned in section 3.5.3, gyroscopes have a zero-offset that is measured during initialization and it was attempted to measure offline, but failed because that offset changes with temperature and will change during usage that is not currently accounted for. A way to more accurately calibrate the gyroscope is to measure the offsets at a range of temperatures and during operation dynamically compute the offset, like proposed in paper [37]. The *MPU*9250 sensor also packs a temperature sensor so it is still a suitable choice. Another approach to tackle the gyroscope drift can be using other sensor fusion algorithms, like the Magdwick filter, that implicitly estimates the gyroscope bias and the magnetometer measurements are optional [29]. Also the different calibration sample sizes hinted that, at a certain sample size, the gains in gyroscope offset estimation accuracy decrease and because the user must stand still during this calibration, if it takes a long time the chances of the user inadvertently move increase, but the impact of the error varies with the magnitude of the movement and because there is more samples, these errors are not as relevant. Also, a very short sample does not estimate the offset with enough precision.

References

[1] B. Fang, D. Guo, F. Sun, H. Liu, and Y. Wu, "A robotic hand-arm teleoperation system using human arm/hand with a novel data glove," 2015. DOI: 10.1109/ROBIO.2015.7419712.

[2] C. Mitsantisuk, S. Katsura, and K. Ohishi, "Force control of human–robot interaction using twin direct-drive motor system based on modal space design," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 4, pp. 1383–1392, 2010. DOI: 10.1109/TIE.2009.2030218.

[3] S. Hirche and M. Buss, "Human-oriented control for haptic teleoperation," *Proceedings of the IEEE*, vol. 100, no. 3, pp. 623–647, 2012. DOI: 10.1109/JPROC.2011.2175150.

[4] T. Ando, R. Tsukahara, M. Seki, and M. G. Fujie, "A haptic interface "force blinker 2" for navigation of the visually impaired," *IEEE Transactions on Industrial Electronics*, vol. 59, no. 11, pp. 4112–4119, 2012. DOI: 10.1109/TIE.2011.2173894.

[5] J. S. Jang and D. Liccardo, "Small uav automation using mems," *IEEE Aerospace and Electronic Systems Magazine*, vol. 22, no. 5, pp. 30–34, 2007. DOI: 10.1109/MAES.2007.365332.

[6] Wikipedia contributors, *Inertial measurement unit — Wikipedia, the free encyclopedia*, [Online; accessed 7-January-2022], 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Inertial_measurement_unit&oldid=1053869659.

[7] W. Geiger, J. Bartholomeyczik, U. Breng, *et al.*, "Mems imu for ahrs applications," in *2008 IEEE/ION Position, Location and Navigation Symposium*, 2008, pp. 225–231. DOI: 10.1109/PLANS.2008.4569973.

[8] M. Cornacchia, K. Ozcan, Y. Zheng, and S. Velipasalar, "A survey on activity detection and classification using wearable sensors," *IEEE Sensors Journal*, vol. 17, no. 2, pp. 386–403, 2017. DOI: 10.1109/JSEN.2016.2628346.

[9] S.-D. Bao, X.-L. Meng, W. Xiao, and Z.-Q. Zhang, "Fusion of inertial/magnetic sensor measurements and map information for pedestrian tracking," *Sensors*, vol. 17, no. 2, 2017, ISSN: 1424-8220. DOI: 10.3390/s17020340. [Online]. Available: https://www.mdpi.com/1424-8220/17/2/340.

[10] Z. Diao, H. Quan, L. Lan, and Y. Han, "Analysis and compensation of mems gyroscope drift," in *2013 Seventh International Conference on Sensing Technology (ICST)*, 2013, pp. 592–596. DOI: 10.1109/ICSensT.2013.6727722.

[11] Y. Akcayir and Y. Ozkazanc, "Gyroscope drift estimation analysis in land navigation systems," in *Proceedings of 2003 IEEE Conference on Control Applications, 2003. CCA 2003.*, vol. 2, 2003, 1488–1491 vol.2. DOI: 10.1109/CCA.2003.1223234.

[12] P. Patonis, P. Patias, I. N. Tziavos, D. Rossikopoulos, and K. G. Margaritis, "A fusion method for combining low-cost imu/magnetometer outputs for use in applications on mobile devices," *Sensors*, vol. 18, no. 8, 2018, ISSN: 1424-8220. DOI: 10.3390/s18082616. [Online]. Available: https://www.mdpi.com/1424-8220/18/8/2616.

[13] R. Hostettler and P. M. Djurić, "Vehicle tracking based on fusion of magnetometer and accelerometer sensor measurements with particle filtering," *IEEE Transactions on Vehicular Technology*, vol. 64, no. 11, pp. 4917–4928, 2015. DOI: 10.1109/TVT.2014.2382644.

[14] W. Chou, B. Fang, L. Ding, X. Ma, and X. Guo, "Two-step optimal filter design for the low-cost attitude and heading reference systems," *IET Science, Measurement & Technology*, vol. 7, no. 4, pp. 240–248, 2013. DOI: https://doi.org/10.1049/iet-smt.2012.0100. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-smt.2012.0100. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-smt.2012.0100.

[15] A. V. Ivanov and A. A. Zhilenkov, "The use of imu mems-sensors for designing of motion capture system for control of robotic objects," in *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 2018, pp. 890–893. DOI: 10.1109/EIConRus.2018.8317231.

[16] H. Chao, C. Coopmans, L. Di, and Y. Chen, "A comparative evaluation of low-cost imus for unmanned autonomous systems," in *2010 IEEE Conference on Multisensor Fusion and Integration*, 2010, pp. 211–216. DOI: 10.1109/MFI.2010.5604460.

[17] N. Y. Ko and T. G. Kim, "Comparison of kalman filter and particle filter used for localization of an underwater vehicle," in *2012 9th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*, 2012, pp. 350–352. DOI: 10.1109/URAI.2012.6463013.

[18] Q. Lam, N. Stamatakos, C. Woodruff, and S. Ashton, "Gyro modeling and estimation of its random noise sources," vol. 5562, Aug. 2003, ISBN: 978-1-62410-090-1. DOI: 10.2514/6.2003-5562.

[19] Q. Li, R. Li, K. Ji, and W. Dai, "Kalman filter and its application," in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, 2015, pp. 74–77. DOI: 10.1109/ICINIS.2015.35.

[20] A. M. Sabatini, "Kalman-filter-based orientation determination using inertial/magnetic sensors: Observability analysis and performance evaluation," *Sensors*, vol. 11, no. 10, pp. 9182–9206, 2011, ISSN: 1424-8220. DOI: 10.3390/s111009182. [Online]. Available: https://www.mdpi.com/1424-8220/11/10/9182.

[21] T. Ainscough, R. Zanetti, J. Christian, and P. D. Spanos, "Q-method extended kalman filter," *Journal of Guidance, Control, and Dynamics*, vol. 38, no. 4, pp. 752–760, 2015. DOI: 10.2514/1.G000118. eprint: https://doi.org/10.2514/1.G000118. [Online]. Available: https://doi.org/10.2514/1.G000118.

[22] A. Cavallo, A. Cirillo, P. Cirillo, *et al.*, "Experimental comparison of sensor fusion algorithms for attitude estimation," *IFAC Proceedings Volumes*, vol. 47, no. 3, pp. 7585–7591, 2014, 19th IFAC World Congress, ISSN: 1474-6670. DOI: https://doi.org/10.3182/20140824-6-ZA-1003.01173. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667016428089.

[23] M. B. Del Rosario, N. H. Lovell, and S. J. Redmond, "Quaternion-based complementary filter for attitude determination of a smartphone," *IEEE Sensors Journal*, vol. 16, no. 15, pp. 6008–6017, 2016. DOI: 10.1109/JSEN.2016.2574124.

[24] R. G. Valenti, I. Dryanovski, and J. Xiao, "Keeping a good attitude: A quaternion-based orientation filter for imus and margs," *Sensors*, vol. 15, no. 8, pp. 19 302–19 330, 2015, ISSN: 1424-8220. DOI: 10.3390/s150819302. [Online]. Available: https://www.mdpi.com/1424-8220/15/8/19302.

[25] ——, "A linear kalman filter for marg orientation estimation using the algebraic quaternion algorithm," *IEEE Transactions on Instrumentation and Measurement*, vol. 65, no. 2, pp. 467–481, 2016. DOI: 10.1109/TIM.2015.2498998.

[26] J. Wu, C. Zhang, and Z. Zhou, "Mav quaternion attitude determination for accelerometer-magnetometer combination: Internal analysis," *tm - Technisches Messen*, vol. 87, no. 10, pp. 647–657, 2020. DOI: doi:10.1515/teme-2019-0158. [Online]. Available: https://doi.org/10.1515/teme-2019-0158.

[27] Z. Liu, W. Liu, X. Gong, and J. Wu, "Simplified attitude determination algorithm using accelerometer and magnetometer with extremely low execution time," *Journal of Sensors*, vol. 2018, p. 8 787 236, Nov. 2018, ISSN: 1687-725X. DOI: 10.1155/2018/8787236. [Online]. Available: https://doi.org/10.1155/2018/8787236.

[28] J. Wu, Z. Zhou, H. Fourati, and Y. Cheng, "A super fast attitude determination algorithm for consumer-level accelerometer and magnetometer," *IEEE Transactions on Consumer Electronics*, vol. 64, no. 3, pp. 375–381, 2018. DOI: 10.1109/TCE.2018.2859625.

[29] S. Madgwick *et al.*, "An efficient orientation filter for inertial and inertial/magnetic sensor arrays," *Report x-io and University of Bristol (UK)*, vol. 25, pp. 113–118, 2010.

[30] R. Mahony, T. Hamel, and J.-M. Pflimlin, "Nonlinear complementary filters on the special orthogonal group," *IEEE Transactions on Automatic Control*, vol. 53, no. 5, pp. 1203–1218, 2008. DOI: 10.1109/TAC.2008.923738.

[31] H. A. Hashim, "Special orthogonal group so (3), euler angles, angle-axis, rodriguez vector and unit-quaternion: Overview, mapping and challenges," *arXiv preprint arXiv:1909.06669*, 2019.

[32] Wikipedia contributors, *Flex sensor — Wikipedia, the free encyclopedia*, [Online; accessed 7-January-2022], 2020. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Flex_sensor&oldid=938952193.

[33] K. Tarchanidis and J. Lygouras, "Data glove with a force sensor," *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 3, pp. 984–989, 2003. DOI: 10.1109/TIM.2003.809484.

[34] J. Cui and Z. Sun, "Visual hand motion capture for guiding a dexterous hand," in *Sixth IEEE International Conference on Automatic Face and Gesture Recognition, 2004. Proceedings.*, 2004, pp. 729–734. DOI: 10.1109/AFGR.2004.1301621.

[35] W. Zhao, J. Chai, and Y.-Q. Xu, "Combining marker-based mocap and rgb-d camera for acquiring high-fidelity hand motion data," in *Proceedings of the ACM SIGGRAPH/eurographics symposium on computer animation*, 2012, pp. 33–42.

[36] Wikipedia contributors, *Calibration — Wikipedia, the free encyclopedia*, [Online; accessed 7-January-2022], 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Calibration&oldid=1054213205.

[37] A. H. Moreira, S. Queirós, J. Fonseca, P. L. Rodrigues, N. F. Rodrigues, and J. L. Vilaca, "Real-time hand tracking for rehabilitation and character animation," in *2014 IEEE 3nd International Conference on Serious Games and Applications for Health (SeGAH)*, 2014, pp. 1–8. DOI: 10.1109/SeGAH.2014.7067086.

[38] T. Seel, J. Raisch, and T. Schauer, "Imu-based joint angle measurement for gait analysis," *Sensors*, vol. 14, no. 4, pp. 6891–6909, 2014, ISSN: 1424-8220. DOI: 10.3390/s140406891. [Online]. Available: https://www.mdpi.com/1424-8220/14/4/6891.

[39] C. Xu, J. He, X. Zhang, X. Zhou, and S. Duan, "Towards human motion tracking: Multi-sensory imu/toa fusion method and fundamental limits," *Electronics*, vol. 8, 2 2019, ISSN: 2079-9292. DOI: 10.3390/electronics8020142. [Online]. Available: https://www.mdpi.com/2079-9292/8/2/142.

[40] S. Zihajehzadeh and E. J. Park, "A novel biomechanical model-aided imu/uwb fusion for magnetometer-free lower body motion capture," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, no. 6, pp. 927–938, 2017. DOI: 10.1109/TSMC.2016.2521823.

[41] J.-O. Nilsson, D. Zachariah, I. Skog, and P. Händel, "Cooperative localization by dual foot-mounted inertial sensors and inter-agent ranging," *EURASIP Journal on Advances in Signal Processing*, vol. 2013, no. 1, p. 164, Oct. 2013, ISSN: 1687-6180. DOI: 10.1186/1687-6180-2013-164. [Online]. Available: https://doi.org/10.1186/1687-6180-2013-164.

[42] C. Xu, J. He, X. Zhang, C. Yao, and P.-H. Tseng, "Geometrical kinematic modeling on human motion using method of multi-sensor fusion," *Information Fusion*, vol. 41, pp. 243–254, May 2018, ISSN: 1566-2535. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253517300301.

[43] S. L. Kilbreath and S. C. Gandevia, "Limited independent flexion of the thumb and fingers in human subjects," en, *J. Physiol.*, vol. 479 (Pt 3), Sep. 1994.

[44] R. V. Vitali, R. S. McGinnis, and N. C. Perkins, "Robust error-state kalman filter for estimating imu orientation," *IEEE Sensors Journal*, vol. 21, no. 3, pp. 3561–3569, 2021. DOI: 10.1109/JSEN.2020.3026895.

[45] R. Campa and H. de la Torre, "Pose control of robot manipulators using different orientation representations: A comparative review," in *2009 American Control Conference*, 2009, pp. 2855–2860. DOI: 10.1109/ACC.2009.5160254.

[46] E. G. Hemingway and O. M. O'Reilly, "Perspectives on euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments," *Multibody System Dynamics*, vol. 44, no. 1, pp. 31–56, Sep. 2018, ISSN: 1573-272X. DOI: 10.1007/s11044-018-9620-0. [Online]. Available: https://doi.org/10.1007/s11044-018-9620-0.

[47] V. Rajeswari and L. P. Suresh, "Investigation and control of principal axes of aircraft using robust method," in *Power Electronics and Renewable Energy Systems*, C. Kamalakannan, L. P. Suresh, S. S. Dash, and B. K. Panigrahi, Eds., New Delhi: Springer India, 2015, pp. 1557–1564, ISBN: 978-81-322-2119-7.

[48] "Anatomy of the hand and wrist bones." (2022), [Online]. Available: https://www.getbodysmart.com/upper-limb-bones/hand-wrist-bones (visited on 07/19/2022).