KU LEUVEN

FACULTY OF ENGINEERING TECHNOLOGY

GROUP T LEUVEN CAMPUS

Open-Source Data Glove Development Kit for XR Applications

Paolo VAN DOMMELEN

Jan RUTRLE

Supervisor:

Prof. Dr. Ir. Carlos Rodriguez-Guerrero

Co-supervisor:

Prof. Dr. MSc. Maria Torres Vega

Master's Thesis submitted to obtain the degree of Master of Science in Electromechanical Engineering Technology

Academic year 2023-2024

ABSTRACT

In an era marked by the increasing adoption of intelligent technology and wearables, the interplay between digital and physical environments has become more tangible. This has further been underlined by the relentless pursuit of research and development in the domains of IoT and wearables in attempts to commercialize ever-more immersive and intuitive interaction technologies within digital environments. Amidst this technological landscape, data gloves have emerged as versatile and immersive interaction technologies, allowing users to engage with digital environments through hand gestures.

Despite these advantages, insights from the literature on commercial data glove technologies have highlighted that challenges such as the lack of open source and developer-friendly solutions and their prohibitive cost, tailored for commercial high-budget sectors, have hindered their widespread adoption. This has resulted in a disparity between commercial and academic developments, pushing numerous researchers to seek alternative solutions offering more open-source and cost-effective solutions.

Based on these identified weaknesses of commercial glove technologies, this thesis aims to bridge the divide between the academic and commercial viability of data glove technology. By developing an open-source Hardware and Software Development Kit (HSDK), namely a framework to prototype, develop, and deploy data glove software and hardware technologies, this thesis seeks to minimize barriers to entry and drive innovation toward more accessible and cost-effective data glove development.

This is achieved by creating a data glove with a focus on cost-effective, open-source, and user-friendly principles, providing hardware and software blueprints accessible via a GitHub repository.

Unlike many commercial products, the hardware of the data glove is made accessible by providing PCB schematics and a set-up of the electronics centered around the ESP32 microcontroller, facilitating the creation of a data glove capable of measuring hand orientation and finger flexions. Additionally, by incorporating popular components and providing additional GPIOs and an EEPROM, the hardware aims to enable the development of configurable and adaptable solutions, simplifying the integration of additional components.

The software architecture enabling communication and visualization of the data obtained from the data glove was developed following similar principles. By implementing high-level visualization software and lower-level middleware that enable user-friendly, flexible cross-platform development and deployment, this software stack aims to provide a versatile development and deployment environment with a straightforward setup while enabling access to third-party services.

In order to validate the system, an experimental analysis was conducted to compare the developed accuracy and precision of the glove at measuring hand orientation and finger flex compared to state-of-the-art data gloves such as the SenseGlove Nova. Through these

methods, it was possible to conclude that the performance of the developed glove is comparable to state-of-the-art. Although the limited time constrained the extent of the development, the open-source nature of all resources holds the promise of further refinement and adoption, positioning this data glove platform as a potential next-generation solution in the field.

ACKNOWLEDGMENTS

Throughout our academic journey, we had the opportunity to study the principles and techniques employed within numerous state-of-the-art technologies. This thesis represents the culmination of our efforts, as our attempt to apply the wide spectrum of software and hardware design techniques we have learned to develop technology that approaches the standards set by engineering teams in the market. While our work as a two-person team may not match the scale and quality set by devices on the market, the realization of this dream was only made possible through the support of those around us.

Firstly, we wish to extend our gratitude to our promotor, Prof. Dr. Ir. Carlos Rodriguez, for his guidance, support, and kindness throughout this journey. His insights, encouragement, and availability were instrumental in shaping our research and aligning it with our personal interests while keeping us inspired and motivated. Moreover, we are grateful for his assistance in evaluating experimental results and for facilitating connections with the staff of the Robotics, Automation, and Mechatronics departments.

We would also like to extend our sincere appreciation to Prof. Dr. MSc. Maria Torres Vega, our co-promotor, for the availability and guidance, which were crucial in shaping the direction of the thesis. Additionally, we are grateful for the opportunity to utilize the SenseGlove Nova for our experimental setup, which significantly contributed to the validation of our results.

Additionally, we are indebted to Marlon Rodriguez Aparicio for his support and availability and for generously providing access to the visual tracker, a key component of our experimental setup.

Lastly, we want to express our heartfelt gratitude to friends and family for their continued support throughout this journey, which has made achieving an ambitious topic such as this one possible.

Paolo van Dommelen

Jan Rutrle

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

φ	Roll – rotation about X	[rad - deg]
ϑ	Pitch – rotation about Y	[rad - deg]
ψ	Yaw – rotation about Z	[red - deg]

LIST OF ABBREVIATIONS

VR	Virtual Reality
AR	Augmented Reality
XR	Extended Reality
EKF	Extended Kalman Filter
HSDK	Hardware and Software Development Kit
SDK	Software Development Kit
EKF	Extended Kalman Filter
MR	Mixed Reality
ML	Machine Learning
NN	Neural Network
IoT	Internet Of Things
HMD	Head Mounted Display
GUI	Graphical User Interface
API	Application Programming Interface
Win	Windows Operating System
OS	Operating System
UI	User Interface
M2M	Machine to Machine
BAN	Body Area Network
PAN	Personal Area Network
LAN	Local Area Network
CAN	Corporate Area Network
MAN	Metropolitan Area Network
WAN	World Area Network
ESP-IDF	Espressif IoT Development Framework
UX	User Experience

1 INTRODUCTION

1.1 Context

With the increasing digitalization of our lives and the continued adoption of smart devices such as smartphones, smartwatches, and smart glasses, the boundary between digital and physical reality continues to blur [1]. As these smart technologies evolve, the demand for more immersive and intuitive interaction technologies has continued to grow, bringing about consistent research and development in the fields of IoT wearables, Virtual Reality (VR), and Augmented Reality (AR) [2], [3]. Among these technologies, touchscreens have become the predominant method of interaction with wearable devices, as can be seen from smartwatches, although they have found less success in applications requiring stronger immersion [3], [4], [5]. This has led to controllers, camera-based interaction methods, and data gloves being widely integrated into VR, AR, and XR applications, revolutionizing multiple industries, from gaming and entertainment to medicine and engineering [3], [4], [5], [6].

Among these, data gloves, a subset of wearable Internet of Things (IoT) devices, are at the apex of this trend toward higher immersion technologies [7]. By translating physical hand and finger movements to digital environments through sensors and motion transducers, users can interact with objects in virtual spaces as naturally as they would in real life, resulting in strong immersion [7]. Due to the interesting prospects of the technology, data gloves have witnessed continuous improvements aimed at meeting the standards set by consumers [6], [7], [8], [9].

Despite their advantages leading to higher immersion in digital environments and their implementation across several industries, data gloves have, over time, failed to reach mainstream markets and have become a less popular option compared to alternatives such as controllers and camera-based digital interaction methods [6]. This can be found in traditional VR setups, which primarily rely on handheld controllers as a gamified interface despite detracting from the immersive experience by imposing a physical barrier between users and the virtual world [3], [4], [5].

Despite the lack of adoption, with increasing interest in these immersion technologies, developments in head-mounted displays (HMDs), among other auxiliary XR technologies, have enabled commercial data gloves to provide increasingly immersive digital experiences [6], [9]. With technologies such as tactile and force feedback, data gloves are able to offer levels of interaction fidelity that standard controllers and other interaction technologies cannot match [6].

1.2 Problem Formulation

Despite their potential, several challenges have hindered the broader adoption of data gloves in professional and, more importantly, consumer markets.

Firstly, while providing numerous tools for cross-compatibility with third-party systems such as APIs and plugins, many data gloves are developed using proprietary communication and hardware solutions, restricting flexibility and developer-friendliness. This has resulted in advancements in research towards open-source and more accessible solutions, as shown by Kristina Grifantini in “Accel Glove” [10], [11].

Additionally, the high cost of commercial data gloves has limited their accessibility primarily to well-funded enterprises and specialized professionals who require high-precision and reliable hand and finger tracking [12], [13]. Therefore, numerous papers by Youngmo Han et al. [13] and Julien Maitre et al. [14], among others, have attempted to provide more consumer-friendly, lower-cost alternatives with similar capabilities.

Ultimately, despite their built-in support for popular VR platforms and top-of-the-line technical specifications, these disadvantages have prevented commercial data gloves from widespread integration. At the same time, this has led to a disconnect between commercial and research/academic-driven data gloves, pushing researchers and consumers alike to seek alternatives better suited to accomplish their goals.

The following sections aim to provide relevant background information on commercial and academic developments in data glove technologies. By identifying market and research gaps from the current state-of-the-art data gloves, this chapter aims to identify an appropriate method for solving the above-mentioned problems that have constrained data glove development.

1.3 State-Of-The-Art Data Gloves

1.3.1 Commercial Data Gloves

1.3.1.1 Overview

Despite the slow adoption, the landscape of data glove technologies has evolved significantly, as shown by Manuel Caeiro-Rodríguez et al. [6], driven by advancements in sensor technologies, computational methods, material technologies, and hardware design [8]. Current state-of-the-art data gloves employ various sensor technologies and haptic and force feedback mechanisms alongside extensive visualization and data acquisition software solutions.

This chapter thoroughly examines the hardware and software technologies used in various commercial data gloves, aiming to provide a clear and comprehensive overview of their key features, technologies, and specifications. Following the discussion from Mingzhang Pan [8], the overview will be conducted across four separate categories: Hardware, Software,

Communication, and Algorithms. Among the many data gloves listed in discussions by Manuel Caeiro-Rodríguez [6], by Bowen Ji et al. [15], and from sources [8], [9], [16], [17], [18], an overview of a subset of state-of-the-art data glove technologies (shown in Table 1) and their features is presented in subsequent tables Table 1, Table 2 and Table 3.

1.3.1.2 Hardware Specifications

Thanks to the insights of Mingzhang Pan [8], it becomes apparent that data gloves employ diverse hand orientation and finger tracking techniques. Thanks to technologies varying from IMUs to bending/flex sensors and fiber optic-based methods, data gloves can accurately predict hand and finger movements with advanced sensor fusion algorithms such as EKFs [8]. Following the analysis conducted by Manuel Caeiro-Rodriguez [6], a selection of hardware components of the identified data gloves from Table 1 can be compared to gain insight into the diverse set of sensor and hardware technologies on the market. These results have been summarized in Table 2.

Table 1:Comparison of Commercial Data Gloves: Pricing, Origin, and Specifications [6], [15], [16].

Glove Name	*Price*	*Weight*	*Battery*	*Year*	*Manufacturer*
5DT Ultra 5/14 [6] [15]	1990-10990	300g	Battery Pack 8h	2020	5dt.com
SenseGlove Nova [6] [15] [16]	5347$ [4999EUR]	315g	Li-Ion Battery 2h	2021	senseglove.com
Manus Prime II Haptics [6]	1499$	60g	Battery 5h	2020	manus-meta.com
Manus Prime X Haptic VR [15] [16]	6173$	70g	Battery 5h	2021	manus-meta.com
Manus Quantum Metagloves [15],	7392$	138g	Battery 4h	2022	manus-meta.com
Rokoko Smartgloves [6] [15]	1245-1495	70g	External Battery Pack 6h	2022	rokoko.com
VRFree v2 [6]	820$ [CHF750]	40g	Battery Pack 24h	2020	sensoryx.tech

Table 2: Comparison of Data Glove Technologies: Tracking Capabilities and Feedback Mechanisms [6], [15], [16], [19], [20], [21], [22].

Glove Name	*Hand Pose Tracking*	*Finger Tracking*	*Feedback*
5DT Ultra 5/14 [6], [19]	N/A	Fiber-Optic	N/A
SenseGlove Nova [6], [16]	IMU,	Rotation encoders	Force & Tactile Feedback
Manus Prime II Haptics [6]	IMU	IMU	Tactile Feedback
Manus Prime X Haptic VR [16], [20]	IMU	IMU + Resistive Flex Sensors	Tactile Feedback
Manus Quantum Metagloves [15], [21]	IMU	(Electro-Magnetic) EMI finger tracking	N/A
Rokoko Smart Gloves [6], [22]	IMU	IMU	N/A
VRFree [6], [20]	N/A	Resistive Flex Sensors, IMU	N/A

1.3.1.3 Sensor Fusion Algorithms

To improve the accuracy of finger and hand orientation tracking, techniques such as sensor fusion algorithms have been widely adopted to run either on microcontrollers or host OSs. Mingzhang Pan [8] and Jie Li et al. [23] provide an overview of different available algorithms for data gloves. While showing that accuracy can be enhanced through calibration techniques involving Neural Networks (NN) and Machine Learning (ML) [8], common sensor fusion algorithms such as Kalman Filters offer alternative hand orientation and finger flexion estimation methods that can run on low-power microcontrollers [23]. Despite the available information on calibration techniques provided by literature, as alluded by source [6], little information can be found on the exact sensor fusion algorithms implemented in commercial data gloves.

1.3.1.4 Communication and Data Accessibility

In addition to powerful hand orientation and finger estimation algorithms, data gloves employ a diverse spectrum of low latency and low-power communication protocols to capture data effectively. Manuel Caeiro-Rodríguez et al. [6] and Bowen Ji et al. [15] highlight the benefits of wireless communication and offer insight into the communication protocols employed by commercial data gloves described in Table 1. These are compiled in the following table:

Table 3: Data Glove Communication Protocols and Developer Access Overview [6], [15], [18], [19], [24], [25].

Glove Name	*Wired Communication Protocol*	*Wireless Communication Protocol*	*Access*
5DT Ultra 5/14 [6], [19]	USB, RS232	Bluetooth	C++ API
SenseGlove Nova [6]	USB	Bluetooth	Force & Tactile Feedback
Manus Prime II Haptics [6], [18]	N/A	Wi-Fi	Manus C++ SDK
Manus Prime X Haptic VR [15], [18]	N/A	Bluetooth	Manus C++ SDK
Manus Quantum Metagloves [15], [18]	USB	Bluetooth	Manus C++ SDK
Rokoko Smartgloves [6], [24]	N/A	Wi-Fi	Rokoko Studio
VRFree [6], [25]	USB-C	N/A	Unity Plugin

1.3.1.5 Software Integration: High- and Low-level software

To enable consumers to visualize and utilize the data from the sensors in data gloves, manufacturers often include software packages that integrate their hardware with popular GUI applications such as Unity and Unreal Engine. These visualization platforms, thanks to their extensive toolset for the development of Extended Reality (XR) applications and extensive history within the Gaming industry, have therefore been adopted as “industry standard” for data-glove-related GUIs [26]. Additionally, many of these visualization techniques enable support either through the use of lower-level C# APIs, or solutions with customized plugins designed to facilitate the integration of third-party systems and hardware, such as data gloves or HMDs [6]. However, these APIs may require different implementations depending on the operating system requiring customized solutions. Below is a breakdown of the supported GUI applications and lower-level software, such as APIs, allowing custom development for the data gloves listed in Table 1.

Table 4: Data Glove Integration Capabilities: GUI Applications, APIs, and OS Support [6], [15], [18], [19], [24], [25], [27], [28], [29], [30], [31].

Glove Name	*GUI & Visualization*	*Low-Level Integration and APIs*	*Supported OS*
5DT Ultra 5/14 [6], [19], [27]	5dt simulations, Autodesk MotionBuilder, 3D Max Studio 6, MATLAB, Siemens Jack	C++, (C#) SDK	Win, Linux, macOS
SenseGlove Nova [6], [28]	Unreal, Unity	C++, (C#) API, ROS	Win, Linux, macOS
Manus Prime II Haptics [6], [18], [29]	Manus Core, Unreal, Unity, Autodesk MotionBuilder	Manus C++ SDK	Win, Linux, macOS
Manus Prime X Haptic VR [15], [18]	Manus Core, Unreal, Unity, Autodesk MotionBuilder	Manus C++ SDK	Win, Linux, macOS
Manus Quantum Metagloves [15], [18]	Manus Core, Unreal, Unity, Autodesk MotionBuilder	Manus C++ SDK	Win, Linux, macOS
Rokoko Smartgloves [6], [24], [30]	Rokoko Studio, Blender, Houdini, Unity, Unreal, Maya, Cinema4d, Reallusion, Autodesk MotionBuilder	N/A	Win, Linux, macOS
VRFree [6], [25], [31]	Unity, Unreal	SDK	N/A

1.3.2 Data Gloves in Research

1.3.2.1 Overview

Despite limited commercial success, data glove technologies have captivated academic researchers, resulting in significant developments and studies. Works by Mingzhang Pan et al. [8], [9], Manuel Caeiro-Rodríguez et al. [6], and Bowen Ji et al. [15] highlight the growing academic interest in enhancing virtual interaction technologies. Following suggestions from literature by Mingzhang Pan et al. [8], [9], the components of data glove technologies can be identified as the following domains: Algorithms, Applications, Hardware Design, and Materials. In addition to the insights provided by these reviews, this section will explore further research-driven innovations in the field of data gloves, setting the stage for subsequent comparisons with the commercial data gloves discussed in the previous section.

1.3.2.2 Hardware Developments

Innovative hardware solutions have, over time, significantly impacted data gloves, from achieving sub-millimeter accuracy for finger and hand orientation sensing to minimizing drift and introducing new ways for haptic feedback [8], [9], [15].

Among the many advancements in the field of sensing technologies, optical linear encoders have garnered attention for their increased accuracy compared to the widely adopted flex and resistive bend sensors. This has been shown through papers by Kang Li et al. [32] and Bowen Jia et al. [15].

Additionally, substantial research explored new methods for implementing force and tactile feedback within data gloves, features highly sought after in commercial models. On this matter, Mohssen Hosseini et al. [33] integrated force sensors and DC motors within a data glove to provide force feedback, a system later implemented commercially by the SenseGlove Nova, as described in the previous section.

1.3.2.3 Algorithms

Sensor fusion and calibration algorithms have also represented a critical area of research in attempts to enhance accuracy and minimize drift. While complex NN and ML-based calibration methods have been reviewed in papers by Mingzhang Pan et al. [8], [9], techniques for sensor fusion, such as Kalman Filters, have also been extensively reviewed. Among these, as shown by Ewout A. Arkenbout, Kalman Filters can be integrated to fuse data from a Nimble VR Visual Tracking System and a 5DT data glove in an attempt to provide higher accuracy and better data completeness [19].

Alternatively, algorithms for gesture recognition have also been extensively reviewed, drawing several different approaches. Among them, as shown by Philipp Achenbach et al. [18], hand-shape recognition systems have been developed to encourage nonverbal communication within VR by utilizing a Manus Prime X and an ML model.

1.3.2.4 Open Source, Low Cost, High-Performance Data Gloves

Much research on data gloves also explored the trend toward developing cost-effective data gloves that maintain high performance and flexibility. In attempts to democratize access to advanced VR and gesture recognition technologies, making them available to a broader audience, sources such as [13], [34], [35], [36] attempt to battle the overly expensive commercial data gloves while providing competitive performance.

On the other hand, data gloves developed, for example, by Bor-Shing Lin, have attempted to provide alternative approaches to the proprietary technologies employed by commercial manufacturers. This has pushed developments towards data gloves comprised of state-of-the-art technologies while enabling higher degrees of flexibility and compatibility with, for example, more modular designs.

1.3.2.5 Applied Interfaces and Control

Lastly, as showcased by Mingzhang Pan in sources [8], [9], commercial data gloves have widely been adopted and reviewed as standalone solutions for specific tasks and scenarios.

Among these, “Haptic feedback in virtual reality, rehabilitation for Parkinson’s Disease" by Mylavan Wegen is an example of how data gloves combined with powerful visual interfaces have been integrated to aid individuals in researching medical, social, and educational problems [25], [37].

Additionally, data gloves have been found to perform well within industrial and medical applications in fields such as robotic teleoperation, assembly tasks, and rehabilitation. This has been shown by papers from C. Mizera et al. and Laura Dipietro et al. [20], [38].

These have clearly showcased the flexibility of data glove technology as a solution for interacting with numerous external devices and systems while simultaneously providing the necessary immersion and digital interaction for XR applications.

1.3.3 Market and Research Gaps

Ultimately, thanks to the insights from literature by Mingzhang Pan et al. [8], [9] and Manuel Caeiro-Rodríguez et al. [6], a comprehensive overview of the data glove market from both commercial and academic viewpoints can be obtained. These reviews highlight the complex nature of data glove technology, which, while noted for its immersion and flexibility, appears to face challenges in achieving satisfactory outcomes across all potential applications. Nonetheless, based on the literature, it can be concluded that the primary market for commercial data gloves includes industries that can support high-budget implementations, such as motion capture, simulation, and medical fields [6].

Despite the technological advancements achieved in data gloves, the reviews suggest a perceived gap in the market, particularly concerning the accessibility of these technologies to a wider consumer base caused by proprietary technologies and high costs. Through the discussed literature, it can also easily be ascertained that despite the numerous commercial opportunities, much research has often led to the development of new data gloves. This ongoing trend is due to the inflexibility and proprietary nature of the hardware and software of commercial data gloves.

In summary, the gathered literature indicates that while there is a robust offering of high-performance data gloves for well-funded sectors, challenges such as the absence of open-source and flexible systems limit broader adoption and innovation outside of a few select sectors. While attempts at addressing these issues have been reviewed in the literature, the exploration of avenues for more standardized, open-source, and cost-effective data glove solutions has yet to yield competitive results, enabling this thesis to seek a more appropriate method.

1.4 Objective

1.4.1 Thesis Aim

The aim of this thesis is to bridge the disparity between academic and commercial data gloves by developing a platform that provides blueprints for both hardware and software development. This platform will facilitate easy, cost-effective, and efficient prototyping, development, and deployment of any data glove system, imposing minimal constraints on users irrespective of the application. Ultimately, the aim of this “Development Kit” is to establish a centralized platform to standardize development practices, enhance resource sharing, and foster a collaborative environment within the data glove community.

1.4.2 Open-Source Development Kit and Vision

Building on the identified market and research gaps, this thesis aims to develop a Hardware and Software Development Kit (HSDK), providing developers with a GitHub repository with diverse tools to create specialized data glove platforms. Whether through developing novel algorithms for microcontroller firmware, creating new front-end and back-end systems, or integrating different middleware, the ultimate goal of this HSDK is to enable significant customization and innovation in data glove technologies.

With the HSDK comes the development of a full-fledged data glove, from the design of hardware, such as the PCB and electronics, to the implementation of firmware, communication, and high-level software running on a host Operating System (OS). This thesis will, therefore, delve into all necessary components for the hardware and software architecture requiring evaluation and analysis of High-Level GUIs, Middlewares, Host OSs, Embedded OSs, communication protocols, and sensor fusion algorithms in addition to all necessary electronics and hardware implementations.

Drawing on the design methodologies outlined by Haider Raad [1], employing collaborative tools and modular design principles is crucial for establishing a robust foundation for the HSDK in this thesis. Requirements for design methodologies, as proposed by Haider Raad [1], will be summarized in the following sections to clarify better the scope of the hardware and software components of the HSDK.

1.4.2.1 Usability (U)

As described by Haider Raad [1], Usability encompasses the 'ease of use' of a device in learning and operating the hardware. This concept can be extended to software systems, where user-friendly libraries and tools simplify development and usage, which may greatly affect the impact of the system [1].

1.4.2.2 Compatibility (C)

Compatibility in software development ensures that devices operate seamlessly across various operating systems, communication protocols, and third-party libraries and software. In the context of hardware, compatibility involves the capacity to integrate additional

modules or components without significant redesign, facilitating easy upgrades and expansions [1].

1.4.2.3 Popularity (P)

Popularity in this context assesses the widespread adoption of a technology and approval within both the user community and industry. This factor plays a crucial role in the long-term relevance and “technical obsolescence” [1] of the platform. While popularity for hardware components is reflected in their extended availability to purchase, in software, popularity can be gauged by the size of the developer community and the richness of the ecosystem of libraries and systems.

1.4.2.4 Scalability (S)

Lastly, Scalability examines the capability of the system to expand its functions efficiently. For both hardware and software, this criterion refers to the ability of a specific development stage to transition to the next with minimal work. For software, this may be reflected by the ability to deploy applications to an increasing number of users with minimal difficulty. In contrast, for hardware, this may reflect the adaptability of the system to be suitable for both the development and deployment phases [1].

1.4.3 Hardware Development Goals

Given the dual nature of the HSDK, which contains both hardware and software elements and is open source, it is crucial to define appropriate boundaries. The hardware development aspect of the thesis will focus on creating a scalable design that can easily integrate with various sensors and actuators, making it adaptable for different use cases and data gloves. This will require the selection of several components, such as appropriate sensors, a microcontroller, and necessary modules for storage. An overview of the components of the hardware architecture can be found in Figure 1. Key considerations from the product development phase, such as popularity, documentation, cost-effectiveness, and wireless capabilities, will guide the selection of components and the design of the PCBs, ensuring that the hardware meets functional requirements and long-term development support [1]. Ultimately, a comparison with a state-of-the-art commercial glove will be conducted to validate the choices made and assess the performance of the designed glove. Using an Atracsys fusionTrack 500 visual tracker [39] as ground truth, hand orientation and finger flexion measurements will be taken. By repeating the experiments using a SenseGlove Nova [28], a well-regarded commercial state-of-the-art data glove, this thesis will allow a comparison between the developed HSDK data glove and industry-standard technology.

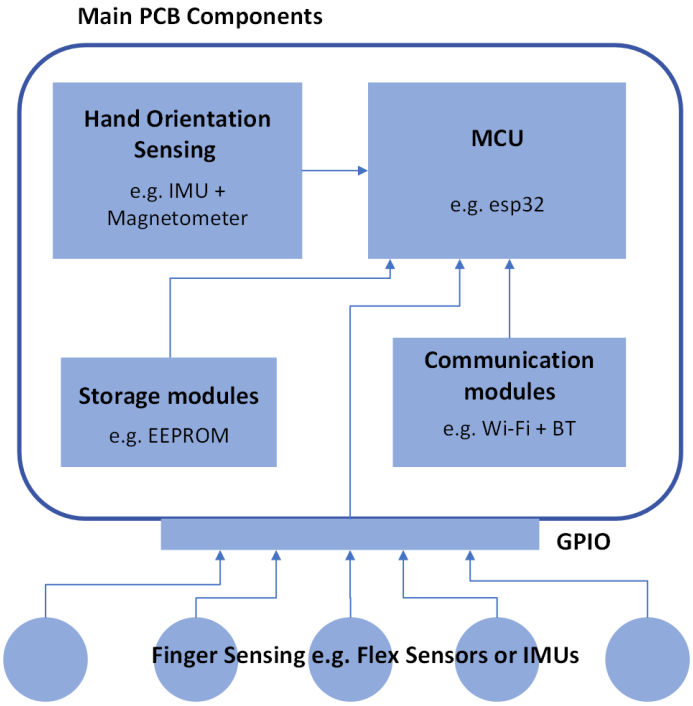

Figure 1: Overview of the Hardware Architecture Components of the HSDK with examples

1.4.4 Software Development Goals

In a similar fashion, the goal of HSDK software is to develop a flexible architecture that can support a wide range of applications, from simple hand tracking to more complex gesture recognition algorithms. This requires the system to provide access to High-Level GUIs or through inter-process communication while enabling communication to the data glove. An overview of the necessary components paper for the final software architecture can be visualized in Figure 2. Following the design considerations proposed by Haider Raad [1], to ensure long-term support and interoperability, the software will utilize popular software development techniques and interfaces, enhanced by robust libraries, to create a powerful development environment. Leveraging tools such as Docker for effective development and deployment environments, ROS2 for its extensive documentation and libraries, and WebSockets for real-time data access, the HSDK will equip developers with a resilient and adaptable framework. This means that, unlike the state-of-the-art data gloves, including SDKs, to port their functionalities to external applications via APIs, this thesis will aim to develop an all-inclusive containerized environment where developers can freely extend and improve with additional libraries and tools. Additionally, by taking advantage of the inherent integration of ROS2 in robotics systems, this architecture aims to be suitable not only for XR applications but also for teleoperation and other industrial settings.

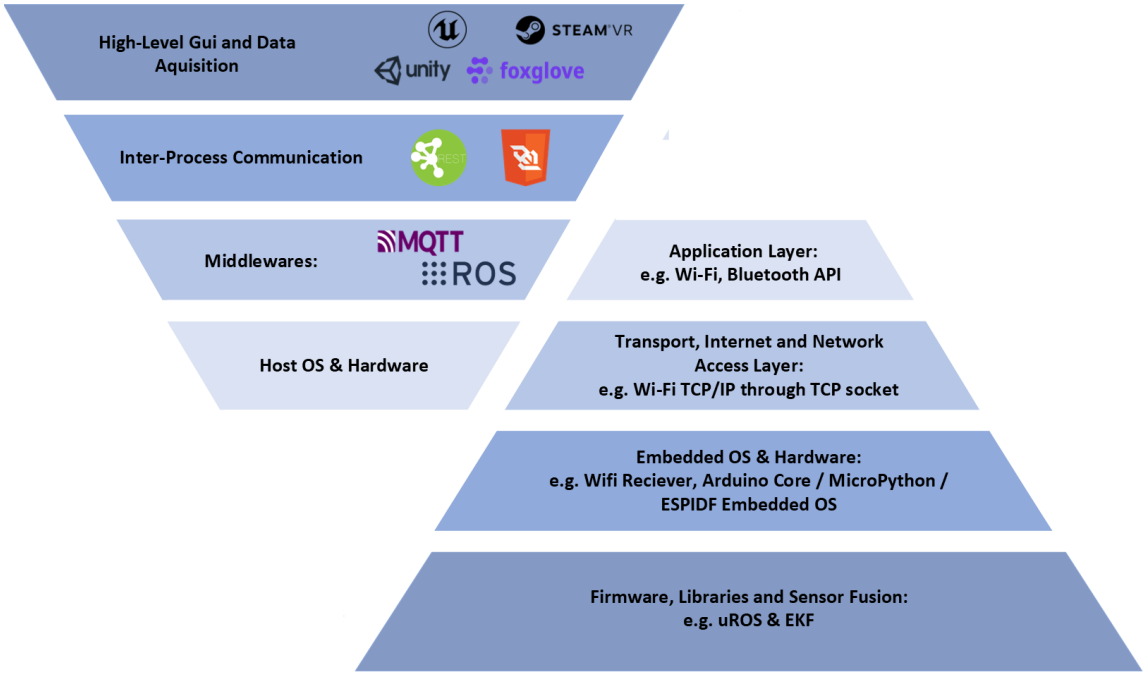

Figure 2: Overview of the HSDK Software Architecture Components and communication interfaces following the TCP/IP model, with example implementations

1.4.5 Thesis Outline

In order to develop the HSDK, this thesis will first delve into a literature analysis of existing technologies in wearables and IoT devices in Chapter 2. By reviewing available orientation and finger-tracking technologies along with a diverse set of communication and software development techniques, Chapter 2 aims to provide the necessary background to back up the design choices made during the development of the Hardware and Software Development Kit (HSDK). Utilizing the insights gained from this literature review, Chapters 3 and 4 will discuss all design choices and implementations made to realize the platform. Specifically, Chapter 3 will discuss the hardware implementations for the data glove, evaluating and discussing the selection and implementation of sensing technologies and sensor fusion algorithms. Similarly, Chapter 4 will analyze the high- and low-level software and communication protocols through a selection and evaluation process and then discuss the implementation of the final architecture. After finalizing the hardware and software of the data glove, Chapter 5 will present conclusions based on experimental comparisons between the developed glove and the SenseGlove Nova, using a fusionTrack 500 tracker. Based on the development process, this paper concludes with Chapter 6, summarizing the results and discussing the future of the HSDK.

2 LITERATURE REVIEW: DATA GLOVE TECHNOLOGIES

2.1 Orientation Tracking Technologies & Challenges

The following section aims to provide the necessary information on orientation tracking technologies, the various types, and their applications based on sources [40], [41], and [42].

2.1.1 Definition of Orientation

Orientation can be defined in multiple ways; however, the three most common representations are the Euler angles, quaternions, and rotation matrix.

Euler angles are three rotations angles, typically denoted as roll φ, pitch ϑ, and yaw ψ, about the principal axis to move from a reference frame to the body frame. Roll is rotation about X, pitch is rotation about Y, and yaw is rotation about Z. While the Euler angles are intuitive to work with, they suffer from the gimbal lock, which occurs when $\vartheta = \pm 90°$, during which the axis of roll and yaw align, resulting in a loss of degree of freedom. Here, the assumed rotation order is Z, Y, X.

This has a secondary effect where the same orientation in the 3D space can be reached with different combinations of the Euler angles, i.e., $\varphi = 90°$, $\vartheta = 90°$, $\psi = 0°$ has the same result as $\varphi = 0°$, $\vartheta = 90°$, $\psi = -90°$. Due to these mathematical shortcomings of Euler angles, quaternions are typically preferred for orientation tracking.

Quaternions do not suffer from these shortcomings, but they are much less intuitive. A quaternion is a 4D vector [*w i j k*], where *i, j, k* are imaginary and *w* is real, and the projection of which onto the 3D space can be used to uniquely define any orientation in the 3D space. As such, quaternions are typically used for orientation tracking.

2.1.2 Orientation Tracking Challenges

Hand orientation tracking presents several different challenges, and as such, each technology offers certain advantages and disadvantages. When considering different technologies, many aspects must be considered for the application of hand orientation tracking.

Accuracy

Accuracy is always desirable; however, increasing accuracy always increases the cost. For example, visual tracking systems are much more accurate than IMUs but come at a much higher cost. As such, it is crucial to find the balance between achievable cost and satisfactory accuracy.

Calibration

The ease at which a technology can be calibrated has an impact on its usability and accuracy. Ideally, a device should track orientation accurately with minimal calibration effort.

Drift

Drift denotes the measuring errors that get summed over time and cause inaccurate results over a longer period. Ideally, the drift should be completely eliminated to ensure proper hand orientation tracking.

Sensitivity to External Factors

Each technology is based on some physical phenomena and, as such, is susceptible to external disturbances. Ideally, the impact of disturbances during hand orientation tracking should be minimized.

Usability in Orientation Tracking

Usability defines how much the tracking device restricts the user. This often must also be balanced with accuracy. The smaller and lighter the device, the lower the accuracy.

2.1.3 Orientation Tracking Technologies

Orientation could be tracked in many ways, but only the most interesting are summarized and considered below.

2.1.3.1 Camera-Based Tracking

Cameras are a popular choice for orientation tracking and position tracking. Their use ranges from the entertainment industry to autonomous cars. Camera-based trackers are typically highly accurate and capable of tracking large numbers of objects and degrees of freedom. However, they also have many downsides compared to alternative methods for hand orientation tracking. They require higher power than other technologies and have low usability as the hand must always be in the view of the camera. They cost considerably more than other approaches and require much higher computational power due to image processing [40], [41].

2.1.3.2 Acoustic-Based Tracking

Acoustic-based tracking makes use of ultrasonic transmitters and receivers. By having multiple receivers and/or transmitters, the hand orientation and position can be determined. While this is a computationally cheaper solution to camera-based tracking, it still does not suffice for a data glove, as components external to the worn glove are required [40].

2.1.3.3 Inertial/Magnetic-Based Tracking

Thanks to advancements in MEMS technology, low-cost IMU sensors can be used for relatively accurate hand orientation tracking. An IMU measures the acceleration along X, Y, and Z, as well as the rotational velocity about X, Y, and Z. Due to the gravity of the Earth, the accelerometer data can be used to determine 2 degrees of freedom, roll, and pitch. Hence, IMUs are typically accompanied by a magnetometer that measures magnetic flux density along X, Y, and Z, which can be used to track yaw due to the magnetic field of the Earth. They are then combined with the gyroscope data via sensor fusion algorithms to better estimate the orientation [40], [42].

This technology has the advantage of being extremely low-cost, lightweight, and small enough to be wearable, and thanks to sensor fusion algorithms, it provides acceptable accuracy for most applications while having zero drift and requiring minimal calibration [40].

2.2 Finger Tracking Technologies & Challenges

Finger tracking is a difficult discipline, and any approach must deal with a number of challenges. Each finger offers very little space for mounting sensors, considering the number of degrees of freedom each finger has. At the same time, any sensor on the fingers must be lightweight and cannot obstruct the user too much. The finger sensors also cannot interfere with the hand orientation tracking sensors, and vice versa. As such, some common technologies are summarized and discussed below.

2.2.1 Camera-based tracking

Similarly to hand orientation tracking, camera-based trackers offer a great advantage in terms of high accuracy and the capability of tracking all the degrees of freedom of each finger. However, the same shortcomings still apply. The fingers must be in the view of the camera at all times. The computational and financial costs are also high compared to other technologies [43].

2.2.2 Inertial-based tracking

The same sensors that can be used for hand orientation tracking can be used for finger tracking. While this technology is a great approach for hand orientation, it is less adequate for finger tracking. A finger offers much less space than a hand for mounting the sensors, resulting in discomfort to the wearer and obstruction of movement. At the same time, the processing cost and time of the whole glove would increase drastically as the microcontroller would need to run six sensor-fusion algorithms in parallel, 1 for each finger and 1 for the hand orientation [44].

2.2.3 Magnet-based tracking

It is also possible to use magnetic fields generated on each finger with magnetic sensors. The magnetic fields can be generated by either a current-conducting ring on the finger or a permanent neodymium magnet. This is a viable option thanks to the small size and low cost. However, one must keep in mind the disturbance that the magnetic fields generated on the fingers create on the magnetometer used for hand orientation tracking [45].

2.2.4 Resistive sensors-based tracking

An alternative approach to finger tracking is the use of resistance-varying sensors. This could be a long strain gage, commonly referred to as a “flex sensor” or a potentiometer. These sensors can track only 1 degree of freedom but are very simple to implement. They offer a great balance between complexity and the amount of information they create [15].

2.3 Communication Technologies & Protocols

This section delves into the communication methods used by microcontrollers in handheld devices, such as Data Gloves, focusing on Machine to Machine (M2M) [46] streaming and capturing data for visualization and post-processing. In order to identify the most suitable communication protocol for the HSDK data glove and to evaluate various possibilities according to device design, usability, and the range of applications, it is necessary to understand the features of a diverse spectrum of wireless communication technologies and their protocols. By providing a comprehensive picture of various IoT communication technologies, this section aims to provide the background knowledge necessary for the discussions of Chapter 4. Sources [1], [47], [48], [49], [50], [51] provide crucial information on various communication technologies used in IoT, wearable devices, and M2M communications and will be used as a reference for the coming sections.

2.3.1 Network Types

A fundamental aspect of the evaluation of communication protocols involves understanding the different communication network types. According to Haider Raad [1], networks in IoT and wearable systems generally utilize a three-layer architecture: devices, gateways, and data centers/cloud. These involve different types of networks, such as Body Area Networks (BAN), Personal Area Networks (PAN), and Local Area Networks (LAN). Before evaluating different protocols, it is crucial to understand the different network architectures employed in M2M communication.

2.3.1.1 Short-Range Network Areas: BAN, PAN

Being the lowest level of communication architecture, BAN and PAN networks usually comprise interconnected devices in the proximity of the user. While Body Area Networks (BAN) strictly include technologies surrounding the body of the customer, such as

wearables and implants for health monitoring, Personal Area Networks (PAN) instead encompass a greater reach of devices within an office room or surrounding environments. Among these networks, as suggested by Veena S. Chakravarthi [49], Wi-Fi, Bluetooth, and Zigbee are the most used forms of communication, allowing for short-range, “over-the-air,” and reliable data transfer [49].

2.3.1.2 Medium-Range Network Areas: LAN, CAN

Medium-range communication architectures, on the other hand, can be classified as Local Area Networks or Campus/Corporate Area Networks. While LAN networks are commonly known to encompass Ethernet and WiFi-based networks, the ability for these communication architectures to be deployed in larger environments with repeaters and extenders comes as a clear advantage to PANs [52].

2.3.1.3 Long-Range Network Areas: MAN, WAN

Metropolitan Area Networks (MANs) and World Area Networks (WANs) similarly indicate the interconnection of multiple LANs over long distances, from metropolitan areas to entire countries, and encompass a wide range of long-distance protocols, such as cellular.

2.3.2 Communication Protocols

To handle communication between networks of wearable and IoT devices, protocols enabling “efficient, secure, and scalable” [1] communication have risen in popularity [1]. Following the classification of network protocols suggested by Haider Raad [1] and Veena S. Chakravarthi [49], this discussion introduces the diverse spectrum of communication protocols and technologies into three primary categories: Long, Medium, and Short-range communication modes. Following the discussions presented by [1], this section presents relevant Physical/Data-Link communication protocols for IoT and wearables, with the goal of understanding their specifications for the analysis conducted in Chapter 4.

2.3.3 Long-Range Communication Protocols

2.3.3.1 Cellular Communication

Among long-range networks, cellular communication offers a broad coverage area and high data rates, making it ideal for applications requiring mobility and significant data transfer, such as common mobile phones. Among IoT technologies, protocols like 4G LTE and 5G have been introduced in emerging massive Machine-Type Communications (mMTC) and could be leveraged further for smart electronics and wearable technologies [53].

2.3.3.2 LPWAN, LoRa, and Sigfox

Low-Power Wide Area Network (LPWAN) technologies, such as LoRa/LoRaWAN, on the other hand, offer a middle ground between the wide coverage of cellular and the low power of short-range networks. These are ideal for sending small amounts of data over long distances (2–5 km in urban areas, 15 km in suburban areas), making them suitable for IoT and wearables requiring low-powered long-distance communication data transmission [1].

2.3.4 Medium-Range Communication Protocols

2.3.4.1 Wi-Fi

Among the medium-range communication networks, Wi-Fi has, over time, become a “staple” and widespread short-range, secure, and high data rate protocol developed on the IEEE 802.11 standard. Primarily deployed in Star and P2P topologies, Wi-Fi can serve devices within a range of approximately 100 meters, offering flexible connectivity without the physical constraints of cables supporting a broad spectrum of applications from streaming media to managing smart IoT environments [1], [50].

2.3.4.2 Ethernet

A popular wired communication protocol, part of the local area networks (LANs) technologies, is Ethernet. While typically considered for wired connections in bus and star topologies, Ethernet supports a wide range of communication speeds, is highly reliable, and allows for numerous variants, including wireless and the support of different protocols [1], [50].

2.3.4.3 Zigbee

Zigbee is a popular suite of high-level communication protocols using small, low-power digital radios. It is particularly effective for creating easy-to-install personal area networks with small, low-power senders/receivers that support mid-range communication up to 300m outdoors. Being a popular choice for Star, Mesh, and Bus topologies, Zigbee is primarily used in mid-range communication scenarios like dada acquisition in automation and medical devices, and other low-power, low-bandwidth needs and could, therefore be a strong choice for wearables [1], [50].

2.3.5 Short-Range Communication Protocols

Short-range technologies like Bluetooth and NFC are perfect for a data glove that requires connectivity to nearby devices without the energy costs associated with cellular or satellite communications. Such technologies primarily include many serial communication technologies detailed by Louis E. Frenzel, Jr [50].

2.3.5.1 Bluetooth

Among the protocols suited for BAN and PAN networks, Bluetooth provides a convenient way to connect and exchange information wirelessly between devices such as smartphones, tablets, and wearables over short distances. Suited for P2P, Star, and Mesh topologies, Bluetooth provides a diverse range of communication speeds and distances, with variants such as BLE being extensively used in IoT and wearables while allowing for secure and encrypted data transfer [1], [50].

2.3.5.2 Near-Field Communication (NFC)

Among the short-range communication protocols, NFC has widely been adopted in smart electronics for short-range P2P communication. NFC primarily enables two devices to

communicate at ranges between 4cm and 20cm, making it ideal for secure, low-power, very short-distance data exchange for applications such as contactless payments and quick device pairing [1], [50].

2.4 Software Architecture

Having discussed the various communication protocols and architectures, it is important to define the components of software architecture aimed at processing and visualizing data from microcontrollers over a network. Similar to the previous chapter, to avoid limiting the selection of software techniques to those offered by the state-of-the-art data gloves reviewed in Chapter 1, this chapter will present a more diverse set of techniques used in IoT, robotics, and wearables. This approach aims to uncover unconventional tools that may better fit the HSDK.

The methods proposed in this chapter aim to offer distinct benefits that would be suitable for the development of an open-source, highly flexible, and cross-compatible software platform. Nonetheless, it is essential to acknowledge that these represent only a snapshot of the extensive range of available software development techniques. This section aims to provide the necessary information about the broad spectrum of such software and libraries, enabling the discussions and conclusions reached in Chapter 4.

2.4.1 GUIs and Visualization Software

2.4.1.1 Unity

Unity is a versatile game engine renowned for its extensive use in creating immersive virtual, augmented, and mixed-reality experiences. It supports a wide range of platforms, including Windows, macOS, and Linux, allowing developers to build and deploy interactive 3D and VR applications seamlessly. The powerful rendering engine of Unity, along with its wide array of plugins, libraries, and tools supporting different forms of hardware and software compatibility, make it a strong choice for developing sophisticated visualization software for IoT devices [6], [54], [55].

2.4.1.2 React & Three.js

As displayed by Boying Wu et al. [56], React combined with Three.js provides an equally powerful framework for building interactive user interfaces and 3D visualizations in web applications. In combination with libraries to enable communication, such as through OPC-UA in the case of Boying Wu et al. [56], the implementation of Three.js allows detailed graphical scenes to be rendered directly in a web browser using WebGL. This combination is particularly powerful for creating accessible and platform-independent IoT applications running on the web, allowing easy development and deployment of GUIs while benefitting from the plethora of tools existing for web development [6], [56], [57], [58].

2.4.1.3 Flutter

Similarly, Flutter, an open-source UI software development kit developed by Google for building cross-platform compatible applications can be employed to develop strong visualization platforms. While not directly tailored towards IoT applications, Flutter, centered around the programming language Dart, offers a strong rendering environment allowing for straightforward, multi-platform development of visualization platforms [6], [57], [59].

2.4.1.4 Steam VR

SteamVR is a full-featured virtual reality hardware and software platform developed by Valve, known for supporting a wide array of VR hardware devices, including various head-mounted displays (HMDs). It provides a rich set of tools for VR development, including motion tracking, input systems, and a vast network for community support, making it ideal for integrating VR capabilities into IoT devices. [60].

2.4.1.5 WebXR

A different approach to VR software applications is offered by WebXR, a Three.js-based API that allows developers to create VR and AR experiences directly for web-based architectures. WebXR, unlike other Web-based GUIs, facilitates the integration of VR experiences integrated into accessible, easy-to-develop and deploy, platform-independent web applications, making it an interesting choice for visualization of data gloves within VR space [58].

2.4.1.6 Qt

Lastly, Qt, a cross-platform development framework, is a different method for developing software for data visualization. By developing high-performance applications suited for desktops, embedded systems and mobile, devices Qt provides numerous advantages as a software development platform suited for IoT and wearable devices' software [61].

2.4.2 Hardware and Visualization

As presented by Andrew Yeh Ching Nee et al. [5], the selection of visualization software for VR/AR/XR/MR is also strongly affected by its hardware compatibility. Over time, this distinction has become ever more pronounced, with software specifically accessible through VR head-mounted displays (HMDs) and many monitor-based visualization systems primarily accessible through the core operating systems Windows, Linux, and Mac OS. It is essential, therefore, to be aware of this additional layer of compatibility [5].

2.4.2.1 Head Mounted Displays (HMDs)

Head-mounted displays (HMDs) are critical in virtual and augmented reality setups. They provide immersive experiences by displaying media right before the eyes of the user. According to the "Springer Handbook of Augmented Reality" [5], HMDs can generally be categorized into "Optical See-Through" and "Video See-Through," each enabling different levels of immersion. The first overlays GUIs and information on transparent lenses, allowing

the virtual objects to be projected onto real objects visually. The latter, on the other hand, presents information on a monitor placed close to the eyes, relaying a processed render of the GUIs and information overlayed onto a low-latency recording from the external cameras. Although both types of HMDs offer unparalleled interaction capabilities and user engagement, they are limited by the requirement for custom-developed virtual environments [5].

2.4.2.2 Monitor-Based Systems

On the other hand, Monitor-based systems continue to be pivotal in applications where direct interaction with virtual environments through VR is unnecessary or impractical. These systems leverage standard display technologies to provide visual feedback and are commonly used in setups where space and cost constraints prohibit the use of more immersive systems [5].

2.4.3 SDKs & Third-Party Access

In order to integrate hardware functionalities and communication protocols within visualization software, a common approach employed by data glove developers, among others, is the development of Software Development Kits [33]. This method, coupled with the integration of their APIs into platform-specific plugins, allows users to facilitate the communication between hardware and software with relative ease. Moreover, it provides users the freedom to integrate the functionalities of the data glove into a visualization software of their choice. These strategies strongly encourage integration with third-party software and utilize various techniques for inter-process communication such as WebSockets, HTTP, REST APIs, and shared memory to enable efficient data transfer.

2.4.3.1 Custom APIs

Custom C# and C++ APIs play a fundamental role in IoT architectures and software development as a whole by allowing tailored functionalities, libraries, and modes of communication to be integrated across various components of an IoT system. These APIs facilitate seamless interactions between software layers and the hardware, enhancing the functionality of the system [62].

2.4.3.2 Platform Specific Plugins

Platforms such as Unity and Unreal Engine strongly support the development of plugins to extend the available set of libraries and tools, further enabling application-specific approaches. Plugins for platforms such as Unity, similar to “Excite-O-Meter” developed by Luis Quintero et al., are crucial for customizing applications to specific needs of IoT devices, facilitating deeper access to hardware capabilities and easier integration of necessary APIs [62].

2.4.3.3 Third-Party Access

Integrating methods for communication with third-party software enables software and IoT applications to benefit from an additional layer of cross-compatibility. Literature on inter-process communication, by Aditya Venkataraman et al. [63], clearly segment such

technologies into Pipes, Sockets, and Shared Memory. Among these, technologies such as WebSockets, HTTP, shared memory, and REST APIs are essential and often implemented data exchange among software applications.

WebSockets provide a full-duplex communication channel over a single, long-lived connection, allowing IoT devices to send and receive data with minimal overhead via protocols such as UDP or TCP. In contrast, shared memory techniques such as simple text streams and XML caching, as employed by Xiaojun Meng [64], enable efficient data transfer between different processes or devices, reducing latency and increasing data throughput [18]. In a similar fashion, the REST API across the HTTP protocol has been shown to work both in IoT environments as an interface between running services [65] and also as a back-end method to integrate with third-party software [65].

2.4.4 Host Operating Systems

Despite the many opportunities that the diverse spectrum of software development techniques offers, it is crucial to consider the effects that different operating systems may introduce. Applications may behave differently across various operating systems, often necessitating tailored development. This means software developers may need to create solutions specific to an individual operating system to harness their full potential, ensuring optimal functionality and performance.

2.4.4.1 Windows, Linux, Mac OS

Windows, Linux, and macOS form the backbone of many IoT systems, each offering robust, versatile, and stable platforms for developing and deploying general software and IoT solutions. The widely recognized Windows and macOS operating systems are known for their broad compatibility with various hardware and software ecosystems. In contrast, Linux is renowned for its security, configurability, and efficient performance in networked environments, making it a top choice for embedded systems, servers, and complex applications [66]. While each operating system provides distinct advantages for developers and their respective user bases, the complexities of cross-platform development have necessitated solutions that abstract and bypass these challenges, ensuring the flexibility needed to meet the diverse requirements of IoT devices and applications [66].

2.4.4.2 Docker

Docker is one of many methods to abstract Operating System restrictions and limitations during software development and deployment. Utilizing an OS-level virtualization, Docker enables applications to be developed and run in containerized environments. Operating in isolated environments from one another, containers bundle their software, libraries, and configuration files, can communicate with one another through well-defined channels, and can be deployed on either Windows, Linux, or macOS thanks to cross-platform compatibility of Docker. This makes Docker a powerful tool in software development and deployment for software and applications for IoT. This way, small containerized applications can be individually tested and scaled easily across a network and wider system [67].

2.4.5 Embedded Operating Systems

After discussing various high-level software techniques, it is crucial to ensure that the utilized microcontrollers are equipped with the necessary tools to communicate and operate effectively with the above-described systems. This section examines several popular embedded operating systems that enable the robust and efficient operation of IoT devices, supporting a range of functionalities from basic task management to complex network communication and real-time processing.

2.4.5.1 Espressif IoT Development Framework (ESP-IDF)

ESP-IDF is the official open-source development framework for the ESP32 and ESP8266 chips, including necessary configurations, source code, and firmware downloaders for the chip, designed for high-performance and low-cost IoT applications. Based on the FreeRTOS, ESP-IDF, through CMake, supports a wide range of development tools and languages, enabling developers to build complex IoT applications that require robust network features and real-time operating capabilities [68].

2.4.5.2 Arduino Core

Similarly, Arduino Core provides an easy-to-use platform for programming the Arduino line of microcontrollers, widely used in education, prototyping, and product development. With plugins supporting additional hardware such as the ESP32 or adding additional functionalities, this framework simplifies the hardware management and coding necessary for IoT applications, making it accessible to broad audiences who benefit from the extensive community support of the platform [68].

2.4.5.3 MicroPython

Based on the programming language, MicroPython is a ”lightweight” implementation optimized to run on a microcontroller, including only a subset of the Python standard libraries. Its ease of use and rapid deployment capabilities make it particularly attractive for prototyping firmware for IoT applications [68].

2.4.6 Middleware & Libraries

Following the discussion on embedded operating systems, additional tools for the software infrastructure can be discussed to provide the system with additional functionalities. Middleware and libraries, providing these extended functionalities, are critical for bridging the gap between operating systems and high-level applications. This section will outline key middleware and libraries to support the development of a robust IoT development platform, enhancing functionality and streamlining application and development.

2.4.6.1 MQTT

MQTT (Message Queuing Telemetry Transport) is a widely adopted middleware, a lightweight messaging protocol for limited bandwidth commonly used in machine-to-machine (M2M) communication and IoT applications. Through a client-broker, publish-subscribe topology, MQTT enables devices to publish real-time topics through periodic messages, facilitating large-scale networks with numerous devices [69], [70].

2.4.6.2 Apache Kafka

Similar to MQTT, Apache Kafka is a distributed real-time publish-subscribe event-streaming and messaging platform. In addition to its popularity, Apache Kafka introduces high throughput, low-latency, multiclient, and reliable communication, ideal for integration in complex IoT ecosystems that require robust data handling [70], [71].

2.4.6.3 ROS/ROS2

The Robot Operating Systems, ROS, and ROS2 are middleware catering to the needs of robotic and IoT device developers by providing a robust framework simplifying the development and deployment of complex concurrent robotic systems. While ROS is primarily deployed on Linux, ROS2 enhances its predecessor through numerous improvements, such as better cross-platform compatibility, including the use of Docker for improved cross-platform development and scalability in deployment. Furthermore, the diverse set of libraries of ROS2 and broad community support further enhance the integration of ROS/ROS2 in IoT and robotic systems by providing flexible and powerful tools for developing and deploying complex systems [72], [73].

2.4.6.4 microROS & Micro XRCE-DDS

Among the many libraries offered by the ROS2 stack, microROS offers a set of libraries and tools for integrating ROS functionalities into microcontrollers, bridging the gap between high-level functionalities and low-level hardware. Interfaceable with both ESP IDF and Arduino Core, microROS extends the robotics and IoT-focused capabilities of ROS into embedded devices while leveraging MICRO XRCE DDS as integrated communication middleware for easy interfacing with other ROS2 devices through a client-agent network [70], [72], [73].

3 HARDWARE

3.1 Orientation Tracking

3.1.1 Component Selection of Orientation Tracking

This section looks at the considerations that go into the selection of core electrical components. The main components that must be selected for a hand orientation tracking glove are the microcontroller, IMU sensor, and magnetometer.

3.1.1.1 Microcontroller

The main criteria when considering microcontrollers are computational speed, communication capabilities, and availability of documentation and other resources. Ideally, the microcontroller should be heavily documented in order to ease the implementation. It also needs to be capable of Wi-Fi and Bluetooth communication, either internally or via an extension by additional modules. It also needs to be fast enough to ensure update frequencies of up to 250Hz. The table below makes a short comparison between the considered microcontrollers.

Table 5: Comparison of Microcontroller Specifications: Clock Speed, Wi-Fi/Bluetooth Capabilities, and Documentation [74], [75], [76]

Microcontroller	*Clock Speed*	*Internal Wi-Fi/Bluetooth*	*External Wi-Fi/Bluetooth*	*Heavily documented*
ATmega328P (Arduino UNO)[74]	20MHz	No	Yes	Yes
RP2040 (Raspberry Pi Pico)[75]	133MHz (without overclocking)	No	Yes	Yes
ESP32-WROOM-32E-Nx [76]	240MHz	Yes	Yes	Yes

Here, the “Internal Wi-Fi/Bluetooth” means the capability is built into the microcontroller and does not require additional chips. “External Wi-Fi/Bluetooth,” on the other hand, implies that the microcontroller can communicate with those channels with the help of additional chips. It is also worth noting that both RP2040 and ESP32-WROOM-32E-Nx have two processor cores, allowing for parallel tasks in case of higher computational loads.

From the table, it is clear that ESP32-WROOM-32E-Nx is the most favorable choice in the context of this thesis as it is a modern microcontroller running at high clock speed and capable of Wi-Fi/Bluetooth communication, saving time and effort in sourcing and implementing additional chips.

3.1.1.2 Inertial Measurement Unit

There are not many considerations that need to be taken into account for the IMU. It merely needs to measure the angular rate and acceleration of the three principal axes at a frequency higher or equal to the update frequency.

A family of sensors that is suitable for this application is LSM6Dxxx by STMicroelectronics. They can sense large ranges of accelerations, up to ±32g, and angular speeds, up to ±2000dps, where g represents the acceleration constant of the Earth and “dps” stands for “degrees per second”. They are able to output data at frequencies of up to 6.66kHz. This family of sensors also offers built-in analog and digital low-pass filters, which can be tuned to increase the accuracy of the hand orientation estimation. Additionally, these sensors are extremely low-cost, costing as little as 1.5 euros [77].

3.1.1.3 Magnetometer

The magnetometer needs to be able to measure magnetic flux density along the three principal axes. Sometimes, it comes built into the IMU; however, the chosen IMU sensor does not contain a magnetometer, so it needs to be selected separately.

A suitable sensor is the MMC5603NJ by MEMSIC. It can provide data at up to 1kHz, has a high detection range of ±0.003Tesla, and a resolution of 6.25 nanoTesla. The magnetic flux density of the Earth is about 20,000 nanoTesla at the equator and 68,000 nanoTesla at the poles, meaning this sensor is more than capable of measuring the changes in the magnetic flux density along its three principal axes as it rotates in 3D space [78] [79].

Similarly to the IMU, it is incredibly low-cost, costing about 0.33Euro.

3.1.2 Sensor-Fusion Algorithms

The domain of fusing gyroscope, accelerometer, and magnetometer data for estimation orientation is very extensive and there exist dozens of algorithms. While they all vary in one aspect or another, they all utilize the three sensors in a similar way.

In this application, the short-term horizon is within 1s, and the long-term horizon is beyond 1s.

A gyroscope is used for short-term estimation of the orientation. Since it outputs the angular velocity, it can be integrated to compute the change in angular orientation; hence, if the initial state is known, all the following states can be determined. However, like any other sensor, it is prone to disturbances and can never be fully accurate, which means that its small errors also get integrated over time, and if it was not combined with an accelerometer and magnetometer, it would drift away from the true orientation over time.

The accelerometer then utilizes the gravity field of the Earth to provide an absolute long-term reference for roll and pitch. While the accelerometer is also susceptible to disturbances and outputs other accelerations due to linear hand movements or centripetal acceleration due to hand rotations, it can be used to prevent drift in roll and pitch.

Magnetometer utilizes the magnetic field of the Earth to provide an absolute long-term reference for yaw. Similar to the accelerometer, it is susceptible to local magnetic disturbances, as such the magnetometer requires calibration to ensure minimal impact on surrounding magnetic fields.

3.1.2.1 IMU calibration

It is possible that the gyroscope and accelerometer come with a bias from the factory. As such, this bias can be measured by placing the sensor on a flat surface and collecting data while keeping it stationary. The values can then be averaged, and the bias computed. The bias can then be subtracted from the future sensor values to minimize the errors.

Naturally, the acceleration along Z should be equal to about 9.81m/s^2 rather than 0 m/s^2, as it is for accelerations along X and Y. The angular rates should then all equal 0 rad/s.

3.1.2.2 Magnetometer calibration

The disturbances on a magnetometer can be split into hard iron and soft iron. The hard iron effect can occur due to manufacturing errors or static surrounding magnetic fields. Soft iron effects occur due to surrounding soft irons that distort the magnetic field around the sensor [80].

The calibration of the magnetometer in this application has been done with the help of MATLAB and its built-in magnetometer calibration function. By collecting data and then passing it into the function, one obtains two matrices, A and B, where A is a 3x3 matrix that corrects for the soft iron effects, and B is a 1x3 matrix that corrects for the hard iron effects [80].

The corrected data then equals to $(raw\ data - B) * A$.

3.1.2.3 Mathematical algorithms

Although there are many variants of mathematical algorithms, this paper will only apply and compare two specific ones.

The first mathematical algorithm chosen is the Extended Kalman Filter. It was chosen because it is a generic and commonly applied sensor-fusion algorithm for many different applications. It combines the sensors based on their relative uncertainties and attempts to compute the best estimate of the orientation [81].

The second mathematical algorithm is the Madgwick filter. It is a much younger algorithm than the EKF and it was developed specifically for fusing gyroscope, accelerometer, and magnetometer data for orientation estimation. It is based on gradient descent and focuses on minimizing computational effort by avoiding expensive operations, such as matrix inversion, present in the EKF, to run efficiently on microcontrollers.

3.1.3 Extended Kalman Filter Implementation

The EKF estimates orientation with two major steps. Prediction is based on the gyroscope output and is followed by the update based on the accelerometer and the magnetometer. This section will present the mathematical equations that were implemented on the ESP32 microcontroller to successfully estimate the hand orientation.

Firstly, the gyroscope data needs to be translated into quaternion rates.

$$\begin{bmatrix} \dot{q}_w \\ \dot{q}_i \\ \dot{q}_j \\ \dot{q}_k \end{bmatrix} = 0.5 * \begin{bmatrix} -q_i & -q_j & -q_k \\ q_w & -q_k & q_j \\ q_k & q_w & -q_i \\ -q_j & q_i & q_w \end{bmatrix} * \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = \vec{\dot{q}}$$

Where $\vec{q}$ is the current orientation in quaternions and $\vec{g}$ are the angular rates in rad/s given by the gyroscope.

The quaternion rates can then be used to compute the prediction of the orientation.

$$\overrightarrow{q_{pred,un}} = \vec{q} + \vec{\dot{q}} * T$$

Where $\vec{q}$ is the hand orientation in quaternions as computed in the previous update cycle, and T is the time step, which is 1/update frequency. Every time quaternions are computed it is beneficial to normalize them to ensure the magnitude remains the same.

$$\overrightarrow{q_{pred}} = \frac{\overrightarrow{q_{pred,un}}}{||\overrightarrow{q_{pred,un}}||}$$

The last step of the prediction step is to compute the new covariance matrix $\boldsymbol{P}$. To compute the new covariance matrix, the Jacobian of $\vec{q}$, denoted as $\boldsymbol{A}$, and the prediction uncertainty matrix $\boldsymbol{Q}$ are required.

$$\boldsymbol{A} = \frac{\partial \vec{\dot{q}}}{\partial \vec{q}} = 0.5 * \begin{bmatrix} 0 & -g_x & -g_y & -g_z \\ g_x & 0 & g_z & -g_y \\ g_y & -g_z & 0 & g_x \\ g_z & g_y & -g_x & 0 \end{bmatrix}$$

The prediction uncertainty matrix $\boldsymbol{Q}$ is a diagonal matrix and practically represents how much “trust” is put into the prediction of each state as compared to the update. In this application, the same value for each state can be used. This matrix can be constant or varying.

When starting, it is best to keep it constant and observe the performance of the EKF. If the orientation estimation has a large error with respect to the real orientation, then the values in this matrix can be adjusted. Whether the error is too large or acceptable depends on the application and user demands.

It is also possible to make this matrix varying. In this thesis, it was found that the magnetometer and accelerometer experience too much disturbance during hand movements; as such, the $\boldsymbol{Q}$ matrix is computed each update cycle according to the angular velocity magnitude.

$$Q = Q_{max} - Q_{slope} * \sqrt{g_x^2 + g_y^2 + g_z^2}$$

$$If\ Q < Q_{min}\ then\ Q = Q_{min}$$

$$\boldsymbol{Q} = \begin{bmatrix} Q_{min} & 0 & 0 & 0 \\ 0 & Q_{min} & 0 & 0 \\ 0 & 0 & Q_{min} & 0 \\ 0 & 0 & 0 & Q_{min} \end{bmatrix}$$

Where Q_{max}, Q_{slope}, Q_{min} are tuned experimentally based on visual feedback and comparison to the ground truth.

The new covariance matrix based on the prediction step is then computed.

$$\boldsymbol{P_{pred}} = \boldsymbol{P} + \left(\boldsymbol{A} * \boldsymbol{P} + \boldsymbol{P} * \boldsymbol{A^T} + \boldsymbol{Q}\right) * T$$

Where $\boldsymbol{P}$ is the covariance matrix as computed in the previous update cycle.

In the update step, the predicted accelerometer and magnetometer readings are computed based on the predicted quaternion orientation and the reference vectors. As explained before, the gravity of the Earth is used as a reference for roll and pitch, and the magnetic field of the Earth is used as a reference for yaw. As such, two reference vectors are created.

$$\vec{G} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\vec{M} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

To predict the accelerometer and magnetometer readings, these two reference vectors must be multiplied with the rotation matrix computed based on the predicted quaternion representation $\boldsymbol{R_O}$.

$$\boldsymbol{R_O} = \begin{bmatrix} q_{pred,w}^2 + q_{pred,i}^2 - q_{pred,j}^2 - q_{pred,k}^2 & 2 * (q_i * q_j - q_w * q_k) & 2 * (q_i * q_k + q_w * q_j) \\ 2 * (q_i * q_j + q_w * q_k) & q_{pred,w}^2 - q_{pred,i}^2 + q_{pred,j}^2 - q_{pred,k}^2 & 2 * (q_j * q_k - q_w * q_i) \\ 2 * (q_i * q_k - q_w * q_j) & 2 * (q_j * q_k + q_w * q_i) & q_{pred,w}^2 - q_{pred,i}^2 - q_{pred,j}^2 + q_{pred,k}^2 \end{bmatrix}$$

The predicted accelerometer and magnetometer readings are then:

$$\vec{h} = \begin{bmatrix} \boldsymbol{R_O^T} * \vec{G} \\ \boldsymbol{R_O^T} * \vec{M} \end{bmatrix}$$

Similarly, as in the prediction step, it is necessary to compute the Jacobian of $\vec{h}$ in order to linearize the system.

$$\boldsymbol{C} = \frac{\partial \vec{h}}{\partial \overrightarrow{q_{pred}}} = 2 * \begin{bmatrix} -q_{pred,j} & q_{pred,k} & -q_{pred,w} & q_{pred,i} \\ q_{pred,i} & q_{pred,w} & q_{pred,k} & q_{pred,j} \\ q_{pred,w} & -q_{pred,i} & -q_{pred,j} & q_{pred,k} \\ q_{pred,k} & q_{pred,j} & q_{pred,i} & q_{pred,w} \\ q_{pred,w} & -q_{pred,i} & q_{pred,j} & -q_{pred,k} \\ -q_{pred,i} & -q_{pred,w} & q_{pred,k} & q_{pred,j} \end{bmatrix}$$

Where $\boldsymbol{C}$ is the Jacobian of $\vec{h}$. The next step is to compute the update uncertainty matrix $\boldsymbol{R}$. This can be done in a similar way as the prediction uncertainty matrix $\boldsymbol{Q}$, except the uncertainty increases with increasing angular velocity.

$$R_{acc} = R_{acc,slope} * \sqrt{g_x^2 + g_y^2 + g_z^2} + R_{acc,min}$$

$$If\ R_{acc} > R_{acc,max}\ then\ R_{acc} = R_{acc,max}$$

$$R_{mag} = R_{mag,slope} * \sqrt{g_x^2 + g_y^2 + g_z^2} + R_{mag,min}$$

$$If\ R_{mag} > R_{mag,max}\ then\ R_{mag} = R_{mag,max}$$

Where subscript “acc” designates accelerometer and “mag” designates magnetometer. $R_{acc,slope}, R_{acc,min}, R_{acc,max}, R_{mag,slope}, R_{mag,min}, R_{mag,max}$ need to be tuned experimentally.

The update uncertainty matrix $\boldsymbol{R}$ is then:

$$\boldsymbol{R} = \begin{bmatrix} R_{acc} & 0 & 0 & 0 & 0 & 0 \\ 0 & R_{acc} & 0 & 0 & 0 & 0 \\ 0 & 0 & R_{acc} & 0 & 0 & 0 \\ 0 & 0 & 0 & R_{mag} & 0 & 0 \\ 0 & 0 & 0 & 0 & R_{mag} & 0 \\ 0 & 0 & 0 & 0 & 0 & R_{mag} \end{bmatrix}$$

The Kalman gain matrix $\boldsymbol{K}$ can then be computed.

$$\boldsymbol{K} = \frac{\boldsymbol{P_{pred}} * \boldsymbol{C}^T}{\boldsymbol{C} * \boldsymbol{P_{pred}} * \boldsymbol{C}^T + \boldsymbol{R}}$$

The next step in an EKF is typically then to compute the new states based on the Kalman gain and the error of predicted and measured sensor readings; however, when it comes to using magnetometers for orientation tracking there is an additional challenge.

The magnetic field of the Earth is not entirely horizontal; instead, it is inclined and, therefore, can be split into horizontal and vertical components [78], [81]. This could be taken into account in the reference vector; however, in that case, the reference vector would change with the geographical location on Earth as the inclination angle is not constant. Instead, the vertical component can be removed from the magnetometer readings by transforming the magnetometer readings into the reference frame using the rotation matrix. $\boldsymbol{R_O}$, setting the magnetometer reading along Z to 0, normalizing the subsequent magnetometer values, and transforming them back into the sensor body frame.

$$\overrightarrow{m_{un}}' = \boldsymbol{R_O} * \vec{m}$$

Where $\vec{m}$ is the magnetometer output in microTesla.

$$m'_{un,z} = 0$$

$$\overrightarrow{m'} = \frac{\overrightarrow{m_{un}}'}{||\overrightarrow{m_{un}}'||}$$

$$\overrightarrow{m_{body}}' = \boldsymbol{R_O^T} * \overrightarrow{m'}$$

The accelerometer output also needs to be normalized.

$$\overrightarrow{a_{body}} = \frac{\overrightarrow{a_{un}}}{||\overrightarrow{a_{un}}||}$$

The measurement vector $\vec{y}$ can then be created, and the quaternion orientation can be updated.

$$\vec{y} = \begin{bmatrix} \overrightarrow{a_{body}} \\ \overrightarrow{m_{body}}' \end{bmatrix}$$

$$\overrightarrow{q_{un}} = \overrightarrow{q_{pred}} + \boldsymbol{K} * (\vec{y} - \vec{h})$$

$$\vec{q} = \frac{\overrightarrow{q_{un}}}{||\overrightarrow{q_{un}}||}$$

Lastly, the covariance matrix $\boldsymbol{P}$ also needs to be updated.

$$\boldsymbol{P} = (\boldsymbol{I} - \boldsymbol{K} * \boldsymbol{C}) * \boldsymbol{P_{pred}}$$

Where $\boldsymbol{I}$ is a 4x4 identity matrix.

These computational steps are then repeated at a desired update frequency to track the hand orientation. It is also worth noting that the quaternion orientation vector $\vec{q}$ and the covariance matrix $\boldsymbol{P}$ needs to be initialized at random values since the starting orientation is unknown. However, thanks to the absolute reference provided by the accelerometer and magnetometer, the orientation estimation converges with the measurement error within a few seconds.

3.1.4 Madgwick Filter Implementation

The Madgwick filter estimates orientation via gradient descent, which avoids computationally expensive operations such as the inversion of a matrix. This section goes over the equations that were practically implemented on the microcontroller to estimate orientation successfully. The equations are based on the original academic paper that derives the Madgwick filter and practical microcontroller implementations that were made available publicly [82].

Same as with the EKF, the first step is to compute the quaternion rates based on the gyroscope data.

$$\begin{bmatrix} \dot{q}_w \\ \dot{q}_i \\ \dot{q}_j \\ \dot{q}_k \end{bmatrix} = 0.5 * \begin{bmatrix} -q_i & -q_j & -q_k \\ q_w & -q_k & q_j \\ q_k & q_w & -q_i \\ -q_j & q_i & q_w \end{bmatrix} * \begin{bmatrix} g_x \\ g_y \\ g_z \end{bmatrix} = \vec{\dot{q}}$$

Where $\vec{q}$ is the current orientation in quaternions and $\vec{g}$ are the angular rates in rad/s given by the gyroscope.

The accelerometer and magnetometer data then need to be normalized.

$$\overrightarrow{a_{body}} = \frac{\overrightarrow{a_{un}}}{||\overrightarrow{a_{un}}||}$$

$$\overrightarrow{m_{body}} = \frac{\overrightarrow{m_{un}}}{||\overrightarrow{m_{un}}||}$$

Where $\overrightarrow{a_{un}}$ is the accelerometer data in m/s^2 and $\overrightarrow{m_{un}}$ is the magnetometer data in microTesla. The magnetic reference vector then needs to be computed.

$$h_x = m_{\text{body},x} * q_w^2 - 2 * q_w * m_{\text{body},y} * q_k + 2 * q_w * m_{\text{body},z} * q_j + m_{\text{body},x} * q_i^2 + 2 * q_i * m_{\text{body},y} * q_j + 2 * q_i * m_{\text{body},z} * q_k - m_{\text{body},x} * q_j^2 - m_{\text{body},x} * q_k^2$$

$$h_y = 2 * q_w * m_{\text{body},x} * q_k + m_{\text{body},y} * q_w^2 - 2 * q_w * m_{\text{body},z} * q_i + 2 * q_i * m_{\text{body},x} * q_j - m_{\text{body},y} * q_i^2 + m_{\text{body},y} * q_j^2 + 2 * q_j * m_{\text{body},z} * q_k - m_{\text{body},y} * q_k^2$$

$$\vec{b} = \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

$$b_x = \sqrt{h_x^2 + h_y^2}$$

$$b_y = 0$$

$$b_z = -2 * q_w * m_{\text{body},x} * q_j + 2 * q_w * m_{\text{body},y} * q_i + m_{\text{body},z} * q_w^2 + 2 * q_i * m_{\text{body},x} * q_k - m_{\text{body},z} * q_i^2 + 2 * q_j * m_{\text{body},y} * q_k - m_{\text{body},z} * q_j^2 + m_{\text{body},z} * q_k^2$$

The gradient corrective step can then be calculated.

$$
\begin{aligned}
s_w = \; & -2 * q_j * \left(2 * q_i * q_k - 2 * q_w * q_j - a_{\text{body,x}}\right) + 2 * q_i \\
& * \left(2 * q_w * q_i + 2 * q_j * q_k - a_{\text{body,y}}\right) - b_z * q_j \\
& * \left(b_x * \left(0.5 - q_j^2 - q_k^2\right) + b_z * \left(q_i * q_k - q_w * q_j\right) - m_{\text{body,x}}\right) \\
& + \left(-b_x * q_k + b_z * q_i\right) \\
& * \left(b_x * \left(q_i * q_j - q_w * q_k\right) + b_z * \left(q_w * q_i + q_j * q_k\right) - m_{\text{body,y}}\right) + b_x * q_j \\
& * \left(b_x * \left(q_w * q_j + q_i * q_k\right) + b_z * \left(0.5 - q_i^2 - q_j^2\right) - m_{\text{body,z}}\right)
\end{aligned}
$$

$$
\begin{aligned}
s_i = \; & 2 * q_k * \left(2 * q_i * q_k - 2 * q_w * q_j - a_{\text{body,x}}\right) + 2 * q_w * \left(2 * q_w * q_i + 2 * q_j * q_k - a_{\text{body,y}}\right) \\
& - 4 * q_i * \left(1 - 2 * q_i^2 - 2 * q_j^2 - a_{\text{body,z}}\right) + b_z * q_k \\
& * \left(b_x * \left(0.5 - q_j^2 - q_k^2\right) + b_z * \left(q_i * q_k - q_w * q_j\right) - m_{\text{body,x}}\right) \\
& + \left(b_x * q_j + b_z * q_w\right) \\
& * \left(b_x * \left(q_i * q_j - q_w * q_k\right) + b_z * \left(q_w * q_i + q_j * q_k\right) - m_{\text{body,y}}\right) \\
& + \left(b_x * q_k - 2 * b_z * q_i\right) \\
& * \left(b_x * \left(q_w * q_j + q_i * q_k\right) + b_z * \left(0.5 - q_i^2 - q_j^2\right) - m_{\text{body,z}}\right)
\end{aligned}
$$

$$
\begin{aligned}
s_j = \; & -2 * q_w * \left(2 * q_i * q_k - 2 * q_w * q_j - a_{\text{body,x}}\right) + 2 * q_k \\
& * \left(2 * q_w * q_i + 2 * q_j * q_k - a_{\text{body,y}}\right) - 4 * q_j * \left(1 - 2 * q_i^2 - 2 * q_j^2 - a_{\text{body,z}}\right) \\
& + \left(-2 * b_x * q_j - b_z * q_w\right) \\
& * \left(b_x * \left(0.5 - q_j^2 - q_k^2\right) + b_z * \left(q_i * q_k - q_w * q_j\right) - m_{\text{body,x}}\right) \\
& + \left(b_x * q_i + b_z * q_k\right) \\
& * \left(b_x * \left(q_i * q_j - q_w * q_k\right) + b_z * \left(q_w * q_i + q_j * q_k\right) - m_{\text{body,y}}\right) \\
& + \left(b_x * q_w - 2 * b_z * q_j\right) \\
& * \left(b_x * \left(q_w * q_j + q_i * q_k\right) + b_z * \left(0.5 - q_i^2 - q_j^2\right) - m_{\text{body,z}}\right)
\end{aligned}
$$

$$
\begin{aligned}
s_k = \; & 2 * q_i * \left(2 * q_i * q_k - 2 * q_w * q_j - a_{\text{body,x}}\right) + 2 * q_j * \left(2 * q_w * q_i + 2 * q_j * q_k - a_{\text{body,y}}\right) \\
& + \left(-2 * b_x * q_k + b_x * q_i\right) \\
& * \left(b_x * \left(0.5 - q_j^2 - q_k^2\right) + b_z * \left(q_i * q_k - q_w * q_j\right) - m_{\text{body,x}}\right) \\
& + \left(-b_x * q_w + b_z * q_j\right) \\
& * \left(b_x * \left(q_i * q_j - q_w * q_k\right) + b_z * \left(q_w * q_i + q_j * q_k\right) - m_{\text{body,y}}\right) b_x * q_i \\
& * \left(b_x * \left(q_w * q_j + q_i * q_k\right) + b_z * \left(0.5 - q_i^2 - q_j^2\right) - m_{\text{body,z}}\right)
\end{aligned}
$$

The corrective steps need to be normalized.

$$\overrightarrow{s_{un}} = \begin{bmatrix} s_w \\ s_i \\ s_j \\ s_k \end{bmatrix}$$

$$\vec{s} = \frac{\overrightarrow{s_{un}}}{||\overrightarrow{s_{un}}||}$$

The quaternion rates can then be corrected.

$$\overrightarrow{\dot{q}_{cor}} = \vec{\dot{q}} - \beta * \vec{s}$$

Where beta is the rate at which the gradient descent algorithm converges. This is the only value that needs to be tuned in this algorithm. The higher the value, the faster the convergence rate and the better the algorithm can track fast hand movements. However, the higher the beta value, the higher the risk of overshooting the minimum and producing an orientation estimate with a higher error. The last step is to update the orientation quaternions, where T is the time step.

$$\overrightarrow{q_{un}} = \vec{q} + \overrightarrow{\dot{q}_{cor}} * T$$

$$\vec{q} = \frac{\overrightarrow{q_{un}}}{||\overrightarrow{q_{un}}||}$$

As with the EKF, these computational steps are repeated every update cycle. The initial quaternion orientation vector also needs to be initialized at a random value, though, as with the EKF, it converges within the measurement error within a few seconds.

3.2 Finger Tracking

3.2.1 Finger Tracking Using Magnets and Hall Effect Sensors

As discussed in section 1.3, tracking fingers is a whole domain on its own and is very complex due to the large number of degrees of freedom. Therefore, in this thesis, the problem was heavily simplified by only tracking the bending of each finger as 1 degree of freedom. While this results in under-tracked finger movements, it still provides enough information for interactive use in extended reality applications.

Since space and movement obstruction are two key critical aspects of finger sensors, the first sensor that was designed was a permanent magnet placed on the finger and a linear hall effect sensor placed on the hand. The analog output of the hall effect sensor could then be linked to a finger angle. This resulted in a very lightweight sensor setup that created minimal discomfort and obstruction. Figure 3 shows the final prototype of this setup.

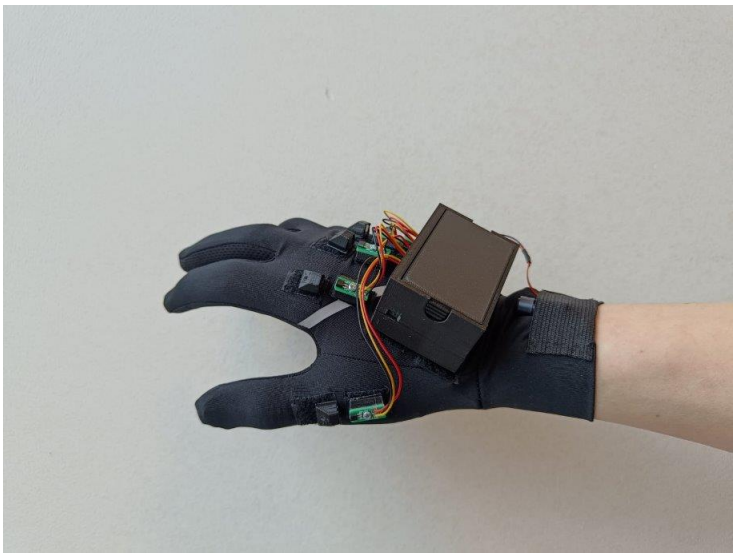

Figure 3: Finger Tracking with Magnets and HES

While this resulted in consistent data that could be used to estimate the finger bending angle, it proved to be unusable with the rest of the glove.

Since a magnetometer is used on the hand to use the magnetic field of the Earth as a reference for yaw, placing five permanent magnets on the fingers creates too much magnetic disturbance, and the magnetic field of the Earth cannot be tracked anymore. As such, orientation tracking is compromised, and this finger-tracking sensor setup cannot be used.

Figures 4 and 5 show how the virtual hand rotates about Z by 180 degrees when the hand with the magnets on the fingers is closed.

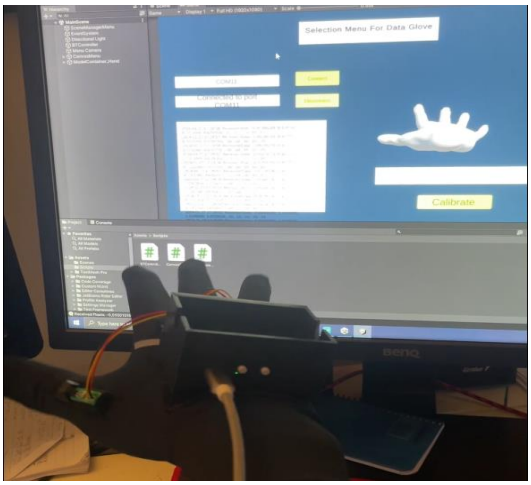

Figure 5: Data Glove using magnetic finger tracking - Hand Open

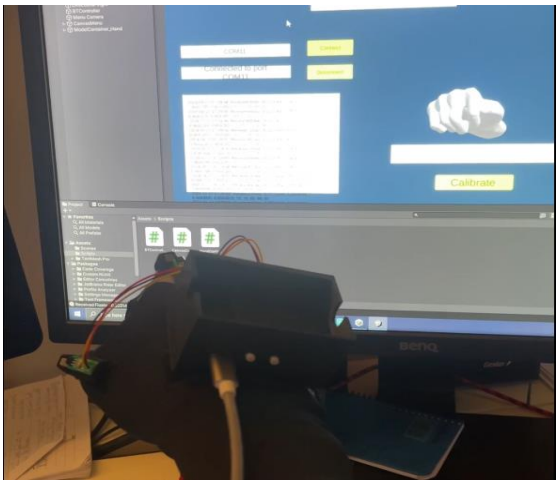

Figure 4: Data Glove using magnetic finger tracking - Hand Closed

3.2.2 Finger Tracking Using Potentiometers

Since magnets could not be used for finger tracking due to the disturbance it caused to the orientation tracking, an alternative way of finger tracking had to be created. Additionally, the new sensor needed to use the same three wires as the hall effect sensors: one wire for 3.3V, one wire for 0V, and one wire for analog voltage signal.

The alternative sensor that was created utilized a tiny potentiometer. This resulted in an electrically simple sensor that outputs consistent data that can be easily linked to the bending angle. Additionally, they do not cause magnetic disturbance on the magnetometer.

On the other hand, it resulted in a mechanically more complex sensor. A string attached to the finger is used to actuate a small lever attached to the potentiometer in one direction, and a spring is used to pull the lever back when the finger moves back up. Figure 8 shows a close-up of the sensor. Figures 6 and 7 show the actuation.

The downside of this sensor is that it requires a more sturdy attachment to the hand and is more uncomfortable to wear. However, it results in an acceptable performance that allows us to gain more information from the glove and provides a base for future developments.

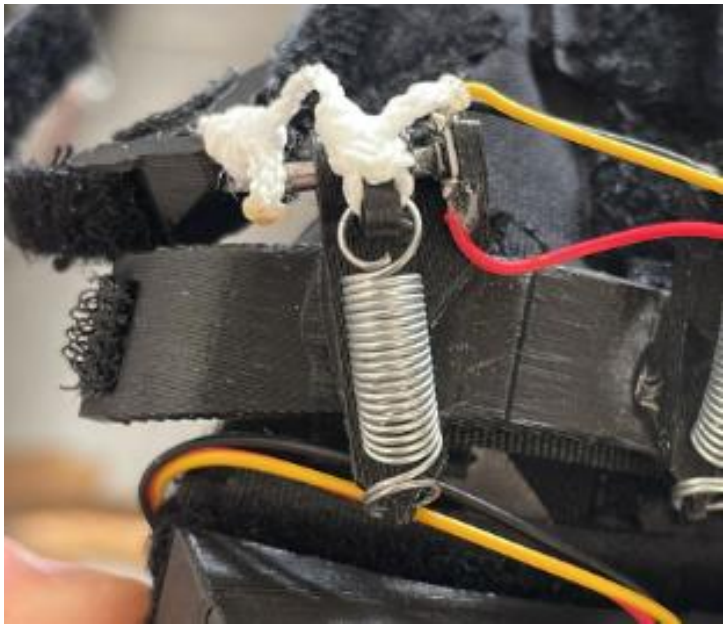

Figure 8: Resistive Finger Tracking using Potentiometers: Hardware Setup

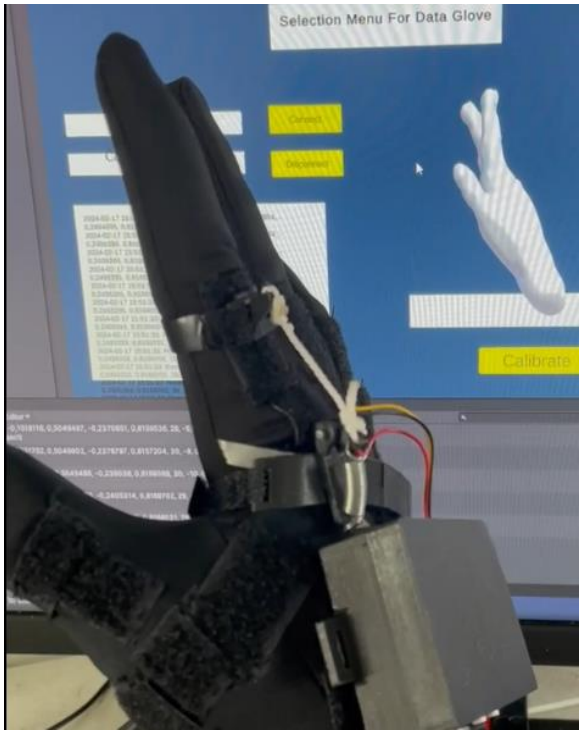

Figure 7: Data Glove using Resistive Finger Tracking - Finger Open

Figure 6: Data Glove using Resistive Finger Tracking - Finger Close

3.3 Final Hardware Architecture

Ultimately, a functional prototype was created that allows for the tracking of the hand orientation and finger movements. With flexibility in mind, the final electronics design allows for easy expansion and application-specific modifications. Figure 9 shows the overall hardware overview. Additionally, a haptic motor drive was added to the existing I2C bus to showcase the ease at which additions can be created. A haptic motor can now connect to the board and provide a simple yet powerful form of haptic feedback, pushing the glove slightly beyond the original goal and showing the advantage of open design.

Another goal was for this glove design to be cost efficient. Since the hardware components were carefully picked to provide excellent performance at affordable cost, the overall prototype material cost for one glove was about 50EUR. About 10EUR was the glove itself, approximately 5EUR were the 3D printed components and miscellaneous, and the largest cost of about 35EUR was due to the PCB production and assembly. Since the largest cost was due to electronics production, that is also the cost that would drastically lower was this manufactured at a larger scale. As such, the estimated commercial cost for a pair of these gloves is between 75EUR and 100EUR, which more than fulfilled the cost goals of this thesis.

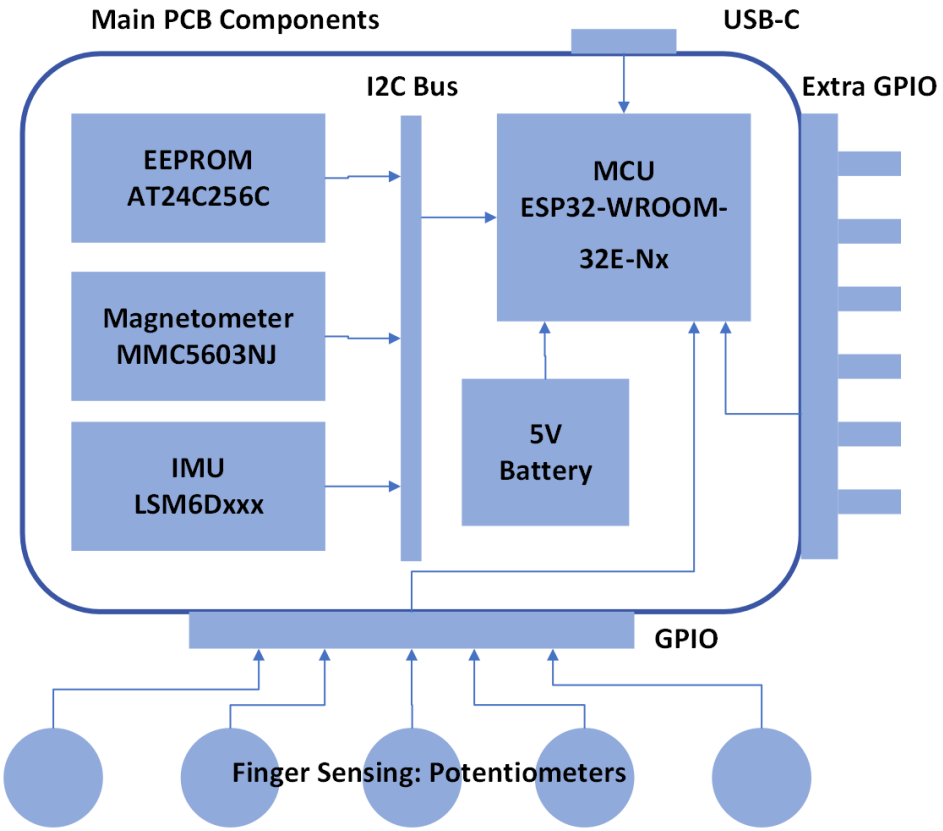

Figure 9: Final ESP-32 Based Hardware Architecture with core components

4 SOFTWARE

4.1 Outline

This section aims to use the findings of Chapter 2 to design the software infrastructure of the HSDK. Given the wide range of applications that the glove developed in this paper aims to satisfy, the software tools and communication protocols identified in Chapter 2 were selected to not only consider techniques widely adopted in existing data gloves but also explore potential IoT alternatives that may offer advantages. Using the techniques as background, this chapter aims to provide both a comprehensive evaluation and selection of the different components for the software architecture but will also discuss the practical implementation and development process of the HSDK. By weighing the specifications of the technologies against the design criteria mentioned in Chapter 1 with factors such as usability and compatibility and the use of evaluation matrices, this chapter aims to choose the set of tools that not only meets the technical requirements for a diverse range of applications but also ensures ease of use and flexibility across different platforms.

4.2 Selection of Communication Protocols

Before evaluating components of the software architecture, this segment explores the diverse communication strategies employed by microcontrollers within handheld devices like Data Gloves, particularly focusing on M2M (Machine to Machine) [46], streaming and capturing data for visualization and post-processing. The primary objective is to identify the most suitable communication protocol for the data glove being developed, considering the impact of various wireless communication technologies and their protocols on device design, usability, and the range of applications.

4.2.1 Communication Technologies and Aim

While it is advantageous to support numerous communication protocols for better compatibility, it can be found through conclusions reached in Chapter 1 and sources such as M. Caeiro-Rodriguez et al. [26] that many commercial data gloves do not support multi-modal communication and often rely solely on protocols such as Bluetooth and Wi-Fi. At the same time, it is advantageous for devices to offer different communication protocols to interface with different systems and hardware. To enable technologies, papers such as [1], [83], [84], [85], [86], [87], [88] give insight into various communication techniques commonly used in IoT, introducing additional possibilities for data communication. The methods identified have been compiled in Table 6.

Table 6: Comparative Overview of Communication Technologies for Data Gloves and IoT Applications [1], [88].

Technology	*Range*	*Data Rate*	*Power Usage*	*Frequency*
USB	< 5 m	1.5 Mbps-10 Gbps	High	Serial
Wi-Fi	<100 m	0.1-54 Mbps	Medium	2.4/5 GHz
RFID	<50 m	0.1-0.6 Mbps	Low	2.4 GHz
Bluetooth/BLE	10-250 m	1-24 Mbps	Low	2.4/5 GHz
ZigBee	10-150 m	20-250 Kbps	Low	2.4 GHz
Cellular 2G/3G	> 100 Km	10 Mbps	High	Cellular
LoRa	1-5 Km	<50 Kbps	Low	Sub-GHz

4.2.2 Evaluation and Decision Matrix

The identified protocols can be easily discerned into three main categories: Long, Medium, and Short-Range. Based on the literature review in Chapters 1 and 2, it can be concluded that long-range protocols, while relevant for specific applications, do not suit the data glove developed in this thesis because of restrictions such as requiring a carrier or specific gateways. On the other hand, as the data glove will operate in either BAN, PAN, or LAN networks, it is essential to carefully consider the appropriate Medium- and Short-range protocols.

Among the identified protocols, it is possible to make a clear distinction between protocols such as ZigBee, Bluetooth Low Energy (BLE), and RFID, namely low-power, low-data rate protocols versus more streamlined high-data rate solutions such as Wi-Fi, Bluetooth, and USB [46], [85], [87], [88]. Following the design criteria identified in Chapter 1, the HSDK focuses on implementing more widely recognized and broadly supported protocols such as Bluetooth, Wi-Fi, and USB. Building on the foundation laid by the literature presented in Chapter 2 and sources [26], [83], [84], it is essential to emphasize the benefits of leveraging popular communication protocols in the development of an open-source hardware and software development kit (HSDK) for the Data Glove covered in this paper. By selecting these well-established communication methods, the aim is for the data glove developed through the HSDK to be versatile and accessible, catering to the needs of a diverse user base, facilitating widespread adoption, and serving a wide range of applications effectively.

While on the one hand, the widespread adoption of Bluetooth, Wi-Fi, and USB for communication owes much to their versatility, popularity, and ability to meet diverse communication needs across various applications [26], [83], their extensive use, as seen in Chapter 3, is equally mirrored in the domain of microcontrollers, where units like the ESP32 come equipped with native support for the three methods [89]. Such built-in functionalities eliminate the necessity for additional external modules, streamlining the design and development process significantly.

This dual-sided adoption not only simplifies the integration of these technologies but also guarantees long-term support for this architecture for devices like the data glove, positioning it as a superior choice for this thesis. Ultimately, the conclusions reached in these discussions can be compiled in the following decision matrix, ranking according to factors identified in Chapter 1, namely Usability (U), Compatibility (C), Popularity (P), and Scalability (S).

Table 7: Decision Matrix - Evaluating Communication Protocols for Data Gloves Using a Scoring System (0-Poor, 1-Below Average, 2-Average, 3-Good, 4-Excellent)

Technology	*Usability*	*Compatibility*	*Popularity*	*Scalability*	*Total Score*
USB	3	4	4	1	12
Wi-Fi	4	4	4	4	16
RFID	3	2	2	1	8
Bluetooth/BLE	3	3	4	3	13
ZigBee	3	3	2	3	11
Cellular 2G/3G	1	1	3	2	7
LoRa	1	2	2	4	9

4.3 Selection of Software Architecture

Following the assessment of communication protocols and their applicability to handheld and wearable IoT devices, this section aims to select the appropriate software infrastructure to add necessary tools and functionalities. Guided by the insights from [87] and [86], a selection of operating systems, development packages, programming languages, libraries, and GUIs will be necessary for shaping the range of applications of the device and user-friendliness as an open-source toolbox. Based on the development goals described in Chapter 1, the aim of the software architecture is to integrate popular software packages in an unrestrained and easily configurable open-source framework, allowing easy adaptation and integration within third-party software.

4.3.1 Higher-Level Architecture: GUIs & Data Access

4.3.1.1 GUIs and Visualization

As demonstrated in the literature presented in Chapter 2 regarding existing data glove platforms and software development for IoT devices, the selection of GUIs and UX design is highly varied and tailored to specific applications [26], [90], [91]. Among the visualization techniques explored, stark distinctions can be drawn based firstly on the hardware compatibility and secondly on the mode of visualization they support, such as VR/AR/MR or simple 2D and 3D monitor-based applications. While platforms like Steam VR have long provided robust development environments for VR applications, the landscape has been

enriched by the emergence of gamified engines tailored for VR/MR/AR development, prominently Unity and Unreal Engine [26]. These engines facilitate the creation of immersive VR environments and support the development of monitor-based 2D and 3D visualization tools with compatibility across various hardware devices. In addition to their expansive libraries and versatility, these industry-standard platforms have garnered widespread adoption across diverse industries, with numerous implementations even from commercial data glove manufacturers [26].

Despite the popularity and advantages offered by these platforms, other tools, less commonly used for data glove development, may offer valuable opportunities. Among these, Android and mobile application-centric approaches such as Vuforia and Flutter [87], Web-VR and web-based GUIs [92], as well as native 2D and 3D interfaces built through Qt [93], [94] offer distinct advantages to 2D, 3D and VR visualization [91]. While allowing developers to create environments in similar ways to Unity and Unreal, platforms such as web-based GUIs, while requiring specific libraries such as OpenGL for 3D visualization or Web-VR for VR visualization, may benefit from the vast set of libraries offered by JavaScript and the straightforward development and deployment pipeline of web applications.

Similarly, while native C++ desktop applications, such as ones built through Qt, may benefit from the vast resources and many libraries offered by C-based environments, they may also drastically increase development complexity compared to standalone software such as Unity, or simpler programming environments for web apps. Despite the numerous pros and cons associated with the software development platforms mentioned above, assessing their compatibility with various hardware platforms is equally crucial when choosing an appropriate visualization method.

4.3.1.2 Hardware Support

As shown in Chapter 2 and the previous section, the choice of visualization software is significantly influenced by the use case and the hardware used. It becomes, therefore, advantageous to implement platforms with a high degree of both hardware and software compatibility. Being industry-standard platforms, Unity and Unreal Engine, while supporting development for both Windows, Linux, and macOS [95], also natively support development for a wide range of “head-mounted displays” (HMDs), such as the Oculus Rift and mobile devices through device-specific plugins [96], [97]. Simultaneously, these platforms offer additional layers of compatibility by enabling visualization through 2D and 3D “Monitor-Based systems,” offering a spectrum of both immersive and non-immersive experiences.

However, this diverse level of adaptability is not found in all mentioned software solutions. While Web-based visualization systems offer the advantage of cross-platform compatibility, they may suffer from hardware restrictions. For example, for web-based interfaces, while HMD and VR-based environments can be tackled through Web-XR and monitor-based “Information Browsers”, can be tackled with WebGL-based Three.js, such methods lack the level of usability and compatibility offered by centralized development platforms such as Unity [92].

While this discussion demonstrates that platforms like Unity provide highly beneficial environments for GUI development that fit the aim of the HSDK, additional considerations must be accounted for regarding how these platforms interface with the glove. As highlighted in Chapter 1, many data glove developers prioritize the implementation of APIs through SDKs over the selection of visualization software. Similarly, in this thesis, to leave the choice of visualization software to the user, the same strategy can be employed. As a consequence, this thesis will not evaluate the various software and instead will ensure the lower-level architecture contains appropriate inter-process communication to interface with these third-party applications.

4.3.1.3 SDKs & Third-Party Access

While understanding the interface between hardware and GUI software is crucial, devices like the Sense Glove Nova 2 and the Leap Motion, along with other devices reviewed in Chapter 1, do not include off-the-shelf direct visualization capabilities within their products [62]. Instead, they offer the core functionalities through a software development kit (SDK), which allows the data to be passed to third-party software via custom APIs for further processing and visualization [62]. This mechanism requires users or developers to implement interface solutions to access and visualize the data effectively, leading to potential interaction errors [62] and added complexity on the side of the user.

This highlights that GUIs do not natively have to be integrated within devices such as data gloves as the SDK and low-level tools integrate with relative ease with numerous third-party software through systems like shared-memory or WebSocket access [98]. Despite the general absence of graphical user interfaces (GUIs) within products such as the Sense Glove Nova 2 [62], it becomes essential to understand how data moves from SDKs and low-level software to higher-level software, ensuring seamless integration within third-party post-processing and data visualization tools.

Hence, it becomes critical when comparing state-of-the-art visualization techniques for VR, AR, MR, and IoT to examine both the methodologies for data interfacing and the range of supported hardware. By considering both the software strategies for data visualization, the hardware compatibility, and ease of integration with third-party software through methods such as WebSocket\Rest [65], the most efficient pathways for developers and users to interface data gloves with and create a flexible, user-friendly environment can be identified. Reflecting on the analysis of visualization techniques and their compatibility with hardware/software, Table 8 presents a concise summary of the primary software tools, detailing their features used in later sections.

Table 8: Comparative Overview of Visualization Techniques for AR/VR and IoT Applications [90], [99], [100], [101], [102].

Example	*GUI Category*	*Deployment*	*Software Integration*	*Hardware Integration*
Unity	3D/2D/AR/ VR	Win/macOS/Linux	C# APIs, Plugins, VR/AR SDKs	Monitor, HMD
React with Three.js	3D/2D	Web-Based	JS, HTML, CSS, APIs (REST/WebSockets)	Monitor
Flutter	2D	Win/macOS/Linux /IOS/ Android	Dart, Firebase, APIs (REST/WebSockets)	Monitor
Vuforia	AR/VR	Win/Linux/macOS /Android	Unity integration, APIs (REST/WebSockets)	Monitor, HMD
Steam VR	AR/VR	Win/Linux	Unity integration, OpenVR SDK	HMD
Qt	2D	Windows, macOS, Linux	C++, Python binding, APIs (REST/WebSockets)	Monitor
Web XR	AR/VR	Web-Based	JavaScript, Web APIs	Monitor, HMD, Projection

4.3.1.4 Evaluation and Decision Matrix

Drawing from the insights gained from the previous sections and the features of different visualization software summarized in Table 2, it can be concluded that embracing various visualization software platforms and hardware types is pivotal for developing wearable IoT device software [86]. This can be achieved by prioritizing systems that ensure ease of integration, whether through SDKs and plugins tailored for specific programming languages and software or through universal protocols like WebSockets. Following the footsteps of manufacturers like SenseGlove, it is possible for the HSDK to sidestep the constraints of direct platform-specific development, favoring Third-Party Software compatibility through WebSocket access, enabling integration across various software environments. However, it is crucial to ensure that such integration is feasible. Therefore, following the results reached through the evaluation matrix in Table 9, the implementation within Unity and simple web-based applications will be tested, ensuring the intended flexibility of the system.

Table 9: Decision Matrix - Evaluating Visualization Techniques for Data Gloves Using a Scoring System (0-Poor, 1-Below Average, 2-Average, 3-Good, 4-Excellent)

Technology	*Usability*	*Compatibility*	*Popularity*	*Scalability*	*Total Score*
Unity	4	4	5	4	17
React with Three.js	3	3	4	3	13
Flutter	3	3	3	3	12
Vuforia	3	4	3	3	13
Steam VR	4	4	4	4	16
Qt	3	3	3	3	12
Web XR	4	4	3	4	15

4.3.2 Lower-Level Architecture: OS & Middlewares

By prioritizing an ecosystem that not only supports bridging to external applications for visualization and integration but also incorporates an intuitive and seamless development experience, selecting the appropriate OS, Software Stack, and Libraries for the open-source HSDK becomes paramount. By evaluating and comparing different components of conventional lower-level software architectures, it is essential to ensure that the framework is both flexible and powerful, capable of meeting the diverse needs of developers and end-users alike for a diverse number of applications.

4.3.2.1 Host OS

In the realm of lower-level architecture, it is crucial to evaluate the diverse operating systems (OS) for the development/deployment of post-processing and visualization software as they bridge lower and higher-level software functionalities, establishing an initial layer of compatibility [103]. Among the Operating Systems (OS) reviewed in Chapter 2, it can be found that, unlike conventional OSs such as Windows and macOS, Docker introduces significant deployment flexibility across various hardware platforms [67]. In this context, Docker acts as a pivotal tool, enabling middleware and SDKs to be utilized within both Linux and Windows inside containers, independently of the host operating system [103], [104]. This versatility is key to the appeal of Docker, allowing users to streamline the development and deployment processes by abstracting away from platform-specific complexities.

4.3.2.2 Embedded OS

Considerations regarding the selection of the appropriate host OS must also account for compatibility with the microcontroller's OS through the communication techniques described in Chapter 2 and at the beginning of this chapter. Using the ESP32 Chip as a reference, Chapter 2 outlines a diverse range of development environments and programming frameworks to introduce a diverse spectrum of approaches to embedded system programming.

Drawing from the insights presented in the literature, it becomes apparent that various microcontroller operating systems offer unique advantages and drawbacks. While on the one hand, platforms powered by ESP IDF present ample opportunities facilitated by CMake and ones supported by MicroPython provide a comprehensive and user-friendly embedded programming environment using Python, both may exhibit differences in functionalities and available libraries [68]. Therefore, it is imperative to evaluate these frameworks not only based on their programming language and environment but also on the supported libraries. This delineates one of the reasons why Arduino Core enjoys widespread popularity.

Furthermore, additional opportunities can be found when evaluating the relevance of middleware such as microROS for embedded system development. While microROS provides a wide spectrum of tools tailored for microcontrollers within the realms of robotics and IoT, many of its functionalities align with the needs of wearable device electronics. Moreover, microROS provides compatibility with the aforementioned ArduinoCore, and ESP IDF embedded OSs [68], [105]. Consequently, by designing the low-level system to accommodate microROS, users can leverage the flexibility of the platform, opting to develop both using either ESP IDF or Arduino Core, enhancing the versatility and usability of the system.

4.3.2.3 Middlewares & Libraries

In addition to a selection of host (e.g., Windows/Linux) and client (e.g., RTOS) operating systems, an additional layer to the lower-level software architecture can be added through the implementation of Middlewares. Following the footsteps of microROS for embedded systems, middleware solutions like ROS/ROS2 provide robust frameworks for robotics and distributed systems but offer ample functionalities for expansions across IoT and wearable applications [106]. Middleware common in IoT, such as MQTT and Apache Kafka [32], can, for example, be found integrated along with ROS in frameworks such as BlazeFlow [69]. BlazeFlow exemplifies the advantages of integrating multiple middleware technologies, as it facilitates efficient data flows across complex, multi-layered systems. The integration of multiple middleware within a “Stack” enables dynamic and flexible system architectures that can effectively support diverse application requirements. Similarly, middleware stacks such as Vulcanexus [107], [108] have integrated ROS2 as the central component of the architecture for its extensive, community-supported libraries which significantly enhance robotic and IoT applications.

Among the wide array of libraries offered by ROS2, ROS Bridge plays a pivotal role by facilitating communication between ROS systems and diverse software environments. The implementation of ROS Bridge as an additional middleware layer would enable the seamless integration of ROS functionalities with web technologies and external applications via WebSockets, further simplifying the creation of user-friendly interfaces and data sharing with high-level software discussed previously in this section [109]. This integration is exemplified by platforms like Foxglove Studio [110], among others, which utilizes WebSockets to connect with ROS, thereby providing a more accessible and interactive experience for users outside the typical robotics sphere [108], [111].

By further accounting for the aforementioned ROS2 native support of the micro ROS middleware [105], it is possible to conclude that the combination of ROS2 and microROS creates a robust environment that enables wearable devices to benefit from the wide spectrum of libraries provided by ROS2 and the support of its extensive community.

4.3.2.4 Evaluation and Selectin

Ultimately, the selection of the components of lower-level architecture is highly dependent on the specific needs of the application and the preference of the developer for programming environments. Each project may require unique features, performance considerations, and compatibility requirements. As the objective of this thesis is to identify a software architecture that is not just robust but also adaptable and flexible to the varied demands of modern robotics and IoT systems, ROS2 stacks such as Vulcanexus [108], [112] stand out with their powerful libraries and tools that cater to a wide range of applications. Furthermore, by abstracting the operating system-related complexities of lower-level development using Docker, it is possible to create an extremely flexible environment that enables development on different operating systems through different embedded development frameworks, enabling easy cross-compatibility using WebSockets. This positions an integration leveraging Docker, ROS2, microROS and ROS Bridge as a flexible and developer friendly environment.

4.4 Proposed Software Stack

Despite the many possibilities brought about by the software tools discussed in this chapter, it is evident from the conclusions drawn in Chapter 1 that companies like Sense Glove incorporate proprietary aspects to their hardware and software design, such as the leveraging of SDKs and custom APIs to encapsulate their low-level functionalities for different operating system [62]. Such proprietary SDKs, while offering streamlined development within specific ecosystems such as Unity [113], often limit cross-platform compatibility and modularity, posing challenges to scalability and integration in diverse environments. Given the review of software tools in this chapter, viable alternatives emerge that offer a pathway to a non-proprietary, flexible, and easily configurable environment, better aligning with the goals of this paper listed in Chapter 1. With all the considerations and conclusions reached in mind, Figure 10 presents a schematic of the final proposed software architecture. Additionally, a higher-level overview displaying the libraries used is presented for additional clarity in Figure 11.

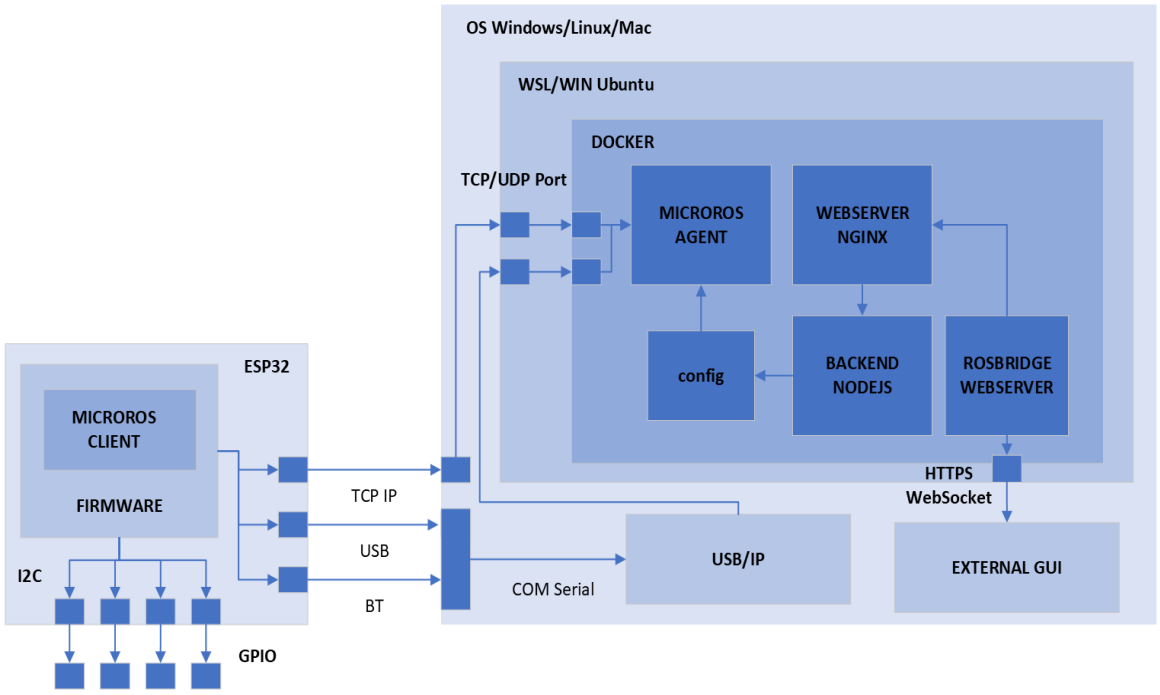

Figure 10: Final High- and Low-level Software Architecture: ESP32 Microcontroller and microROS in a Dockerized Environment

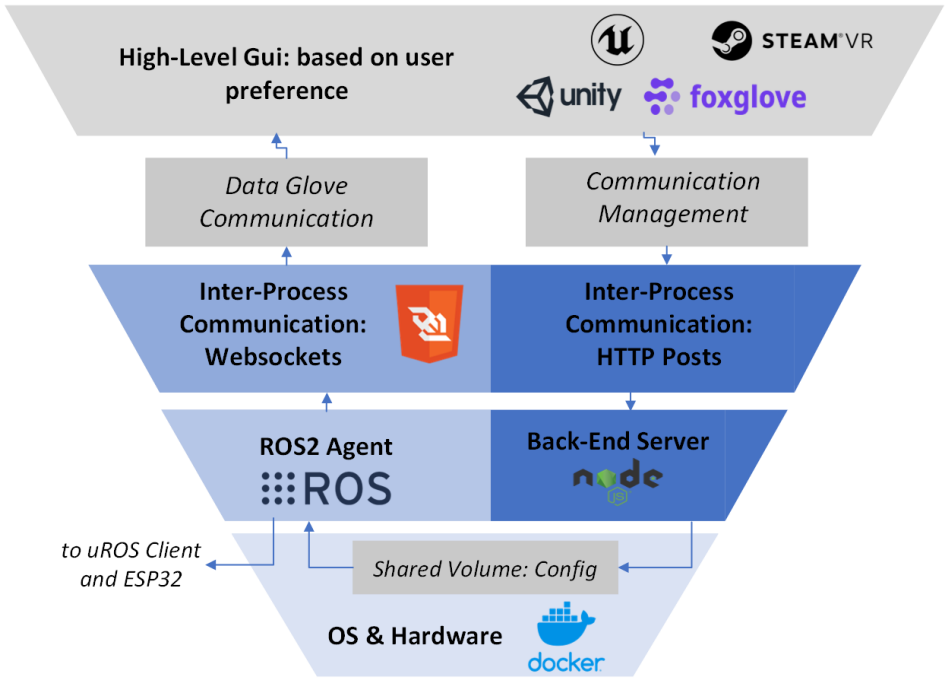

Figure 11: High- and Low-level Software Architecture with emphasis on communication management

4.4.1 OS and Middleware

Based on the conclusions reached in the previous section, the approach taken throughout the development of the lower and higher-level software architecture is centered around the implementation of the ROS2 middleware. While commercial data glove manufacturers, as previously mentioned, implement data glove functionalities in low-level C++ APIs, this paper attempts to provide a more flexible approach by instead implementing a Docker-based development environment. By leveraging the native integration of ROS2 in Linux Docker containers [73], one can, through a Docker image, develop a customized ROS2-based environment with any library of choice and configure multiple containers to operate independently or interactively. This allows for the creation of a highly flexible architecture that can be tailored to specific project requirements without the burden of unnecessary components.

4.4.2 Embedded Systems and Communication

Leveraging ROS2 as core middleware allows the development of the embedded system of the data glove to integrate seamlessly into ROS2-based environments. In addition, microROS introduces seamless integration into the microcontroller to the host network layer, enabling communication through either Wi-Fi, Bluetooth, or USB, as discussed in the previous chapter. Furthermore, as previously mentioned, the microROS library easily interfaces with Arduino Core and ESP IDF-based firmware, ultimately resulting in a flexible and easily configurable framework.

4.4.3 Visualization and Third-Party Software

As established in this chapter, selecting a GUI development framework for HUD-based or monitor-based applications is greatly streamlined by incorporating seamless third-party software access. By configuring the ROS2 Docker container to stream the information of the data glove via a WebSocket on the local network, various visualization software equipped with WebSocket libraries can effortlessly read the data and implement it in custom visualization methods. This eliminates the need for customized plugins and APIs, enhancing the overall usability and flexibility of the development process.

4.4.4 Risks and Weaknesses

While the framework discussed in the preceding sections offers significant flexibility, it is important to recognize potential weaknesses inherent in the system. Firstly, the adoption of ROS2 and Docker introduces the possibility of a more intricate and advanced development environment. Although ROS2 applications can be developed using languages like Python, the utilization of Docker presents opportunities to expand the system in various ways, potentially adding complexity. This complexity stands in contrast to the simplicity offered by APIs utilized by companies like SenseGlove, which facilitate easy interfacing with C++ and C# based applications.

Moreover, it is crucial to acknowledge that while Docker enables the utilization of the ROS2 middleware, it imposes limitations during the deployment phase. Although deployment on virtual machines and Linux-based environments is feasible, the environment developed in this thesis is primarily intended for local deployment (on Linux OS) or continued usage through Docker. While this approach imposes limitations, such as the necessity of Windows Subsystem for Linux (WSL) for running Docker on Windows, this trade-off was considered to ensure ease of configuration of the system and compatibility across Windows, macOS, and Linux platforms.

Lastly, an important weakness of Docker is the lack of an official passthrough from serial COM ports into Docker containers. While this safety feature ensures the isolation of Docker containers from the host machine, it requires USB devices connected to Windows to find an alternative approach to obtain access to the Docker environment. For this reason, this thesis will primarily focus on the implementation of Wi-Fi communication first and foremost and then implement Bluetooth and USB serial communication implementing USB/IP, an additional protocol for sharing serial devices onto the network, allowing them to be accessed from within Docker containers.

4.5 Software Architecture Implementation and Challenges

4.5.1 Introduction to Software Architecture Development

To kickstart the development of the architecture outlined in this chapter, it is crucial to define the minimum viable product (MVP), namely the essential set of functionalities and features needed to achieve the goals of the HSDK. Considering the vast possibilities of software design, this thesis will primarily focus on three main components that constitute the MVP.

Firstly, establishing seamless communication between the host ROS2-based environment and the microROS-based microcontroller is an essential requirement. Secondly, it is crucial to develop an interface capable of effectively monitoring and modifying the microROS communication channel, enabling its opening and closing, enabling change of protocols from Wi-Fi TCP to Bluetooth, and establishing a connection with the data glove through a GUI. Lastly, it is crucial to integrate ROS Bridge to enable the exposure of ROS Nodes to WebSockets for simplified interfacing with third-party software.

As mentioned earlier, the entire framework will be implemented within Docker, utilizing the WSL (Windows Subsystem for Linux) runtime on Windows 11. This choice allows the deployment of services in separate containers, facilitating parallel operation and seamless communication via the local network. The following sections outline the development process of the HSDK and the various components of the software architecture.

4.5.2 ROS2 to microROS Dynamic Interface

While standard configurations of microROS communication within Docker typically employ a static container initialized with fixed port assignments, this approach imposes limitations. To allow users to dynamically adapt communication between the agent and client, a method must be devised to modify the configuration of the agent. Although scripts with Docker privileges can enable dynamic starting and stopping of containers, such methods present significant security concerns. Therefore, a configuration file is implemented, which is automatically read at runtime by the entry point script of the microROS agent to dynamically adapt its configuration. These functions facilitate tasks such as starting and stopping the agent or changing the TCP/UDP port used for communication, enabling the entry point script to modify the microROS agent process dynamically.

With a dynamically changing microROS agent set up on the host machine, it becomes feasible to run the client on the ESP32 microcontroller by flashing firmware using either Arduino Core or ESP IDF. After specifying a fixed TCP port on the network and modifying the configuration to connect to the same port, microROS topics can be read from the microcontroller on the ROS2 Docker image, confirming the establishment of the communication channel.

By implementing a dynamic system modifying the communication between host and client, it is possible to allow users firstly to change protocols of communication but then also implement easily bidirectional communication where from a GUI it is possible to directly send commands to the ESP32 invoking changes to the firmware during usage. While this was not explicitly implemented, it is an intended feature to be evaluated in future revisions.

4.5.3 Front-End Web GUI and Back-End Server

To enhance the capabilities of the dynamically configured microROS agent, a front-end and back-end server architecture has been implemented. This architecture enables commands from a GUI hosted on the front-end web server to be transmitted to the back-end server via HTTPS requests. Subsequently, the back-end server processes these requests and modifies the configuration file accordingly, functioning similarly to a RESTful API server. To facilitate this functionality, the front-end web server is deployed using Nginx Alpine within a Docker container. This lightweight and efficient engine provides a robust platform for hosting the GUI, simplifying the process of sending commands to modify the microROS agent configuration. On the other hand, the back-end server is powered by Node.js 16, enabling it to receive and process incoming POST requests from the front-end GUI, ensuring seamless communication between the GUI and the microROS agent configuration. Through the combination of back and front-end servers, the architecture allows seamless modification to the microROS configuration, enabling a high degree of flexibility in the allowed possibilities.

4.5.4 Docker Compose, Shared Volume, Network Sharing

To implement the entire architecture as one, a Docker Compose file is implemented, including necessary shared volumes and network sharing between containers. Docker Compose streamlines the deployment of the entire system, defining each service within the docker-compose.yaml file and specifying their interconnections. This allows users to modify the system through a comprehensive, configurable list of containers and services. Among these, shared volumes play a crucial role in enabling data sharing between containers. This is especially important in the current setup as it enables storage and access to the configuration file utilized, ensuring services running in the docker environment can access information about microROS communication.

Moreover, the utilization of a custom Docker network enhances communication efficiency between containers. By configuring services to operate within the same network, such as the network defined in the Docker Compose file, containers can communicate with each other seamlessly using container names as hostnames. This is especially beneficial when integrating several ROS2-based services. For example, it is possible for users to implement Teleoperation by utilizing a separate ROS2 container and, at the same time, access the nodes of the data glove through the network. By having the microROS agent running in one container and ROS Bridge running in a different container, data sharing between them is seamless, thanks to the shared network.

4.5.5 ROS Bridge

To facilitate streaming data from the microROS agent container to external applications, a Ros Bridge Server is implemented. Implemented through a Docker image similarly to the other components of the architecture, this server acts as a gateway between ROS2 nodes and WebSockets, enabling seamless data streaming. Like other components, ROS Bridge is containerized using a Docker image. By leveraging the custom Docker network established in the Docker Compose setup, ROS Bridge seamlessly communicates with other ROS2 services running in different containers, allowing ROS2 topics to be accessed by the WebSocket. This interconnected environment enables data access through WebSockets, facilitating data transfer and integration with external applications.

4.5.6 Final Software Architecture

Ultimately, the final software architecture of the HSDK embodies a robust and adaptable framework designed to meet the diverse needs of developers and users in the wearable IoT ecosystem. By integrating cutting-edge technologies and best practices from the fields of robotics, IoT, and software development, the HSDK offers a novel solution for building and deploying wearable IoT applications.

At its core, the architecture leverages ROS2, a powerful middleware for developing complex distributed systems. It provides extensive libraries and community support while enabling seamless communication with microcontrollers. By leveraging the advantages of microROS, client-agent communication is integrated within the software architecture of the

data glove. Despite limitations of microROS in terms of configuration, the architecture introduces a combination of front-end and back-end servers. This integration allows for the dynamic modification of the configuration of the agent, thereby facilitating easy adjustments to the communication setup.

Using a front-end, web-based GUI hosted on an Nginx Alpine server, intuitive tools for monitoring and controlling the data glove can be provided, facilitating real-time interaction and configuration changes. Coupled with a Node.js back-end server, the GUI enables seamless communication with the microROS agent, empowering users to customize and optimize the behavior of the data glove to suit their specific needs.

Additionally, to ensure compatibility across different environments, ROS Bridge was introduced to add the necessary levels of compatibility of the platform. By interconnecting ROS2-based environments with external third-party software through WebSockets, the versatility and compatibility of the architecture are greatly enhanced in accordance with the requirements set in Chapter 1.

At the same time, challenges such as the intricate nature of ROS2 and Docker integration, alongside the limitations posed by Docker during deployment, necessitated careful consideration. While the adoption of ROS2 and Docker offers significant advantages in terms of flexibility and usability, it also introduces complexities in development and deployment. Additionally, the lack of an official passthrough from serial COM ports into Docker containers presents a hurdle in accessing USB devices connected to Windows systems. To mitigate these challenges, the focus was placed on prioritizing Wi-Fi communication and exploring alternative protocols like USB/IP for USB serial communication within Docker containers.

Ultimately, despite complexities introduced by Docker with regards to serial communication passthroughs, the final software architecture of the HSDK offers a novel approach to software development platforms aimed at providing developers a flexible and easily configurable toolset for crafting state-of-the-art data-glove applications. By implementing a Back-End server this architecture enables developers to develop Dockerized applications and manage the communication channel between host and ESP32 easily. This comes as a stark contrast to commercial data gloves that offer proprietary and fixed communication. In addition, by implementing ROS Bridge, the HSDK attempts to draw inspiration from the C++/C# APIs offered by commercial data glove manufacturers to integrate the architecture within numerous third-party software through WebSocket access.

5 EXPERIMENTS

5.1 Method

To validate the performance of the glove and compare it to state-of-the-art gloves, an experiment was conducted. A visual tracking system was used as the ground truth. The performance of the developed glove and a commercial state-of-the-art glove was then measured with respect to this ground truth. The goal is to determine the precision, accuracy, and drift of the glove and how the Madgwick filter and EKF compare in accuracy with respect to each other and the state of the art.

The visual tracking system used was fusionTrack 500 by Atracsys. The commercial state-of-the-art glove used was Nova by SenseGlove. The fusionTrack 500 tracks reflective markers in front of the camera. By using a specific 3D object with a set of markers configured for FusionTrack 500 and attaching it to the gloves, the orientation of the hand in 3D space can be tracked. By using a visual tracking system, the hand movement is completely free, and no complex automated setup is required for moving the gloves.

The hand orientation can then be tracked by the camera and a glove simultaneously. Data is collected in CSV files and timestamps are used to align the data from the fusionTrack 500 and the gloves. While the camera and the gloves output data at different frequencies, it is remedied by using the timestamps to align the data in time.

For each dataset, the glove is first left completely stationary for approximately 60 seconds and then picked up by hand, at which point the glove can be rotated in 3D space while both the camera and the glove are outputting the orientation data. Since the gloves are moved by hand, the movements are not expected to be exactly the same between datasets.

Similarly, the camera can be used for tracking the finger bending angle. By attaching the 3D object with markers to the index finger, the relative angle can be tracked by the gloves and the visual tracking system simultaneously. Since all the finger sensors are the same, only the index finger was measured.

The orientation is compared in Euler angles representation as they are intuitive to work with and compare. 5 sets of data were collected with the Madgwick filter active, where each set had a different beta value. The beta values used were 1.0, 1.25, 1.5, 1.75, 2.0. A set of data was collected with the EKF active where the EKF parameters had the following setting: $Q_{max} = 5, Q_{slope} = 7.5, Q_{min} = 0.01, R_{acc,slope} = 0.0005, R_{acc,min} = 0.001, R_{acc,max} = 0.5, R_{mag,slope}10, R_{mag,min} = 0.001, R_{mag,max} = 10$. A set of data was collected using the Nova glove. A set was collected for finger tracking using the HSDK glove, and a set was collected for finger tracking using the Nova glove.

The update frequency of the HSDK glove was set to 200Hz for all datasets and data was collected at a frequency of 25Hz. The data is processed and visualized using Python.

5.2 Data Processing

5.2.1 Processing the Orientation Tracking

The raw Euler angles from the gloves and the fusionTrack 500 cannot be compared directly, as the fusionTrack 500 and the gloves use different reference frames. As described in Chapter 3, the gloves use the Earth frame formed by the gravity and magnetic field as a reference frame. However, the fusionTrack 500 uses its own reference frame defined internally. As such, the Euler angles measured by the gloves need to be moved from the Earth reference frame to the camera reference frame. The Euler angles can be moved by first converting the Euler angles to orientation matrices.

$$R_O = \begin{bmatrix} \cos(\psi) * \cos(\vartheta) & \cos(\psi) * \sin(\vartheta) * \sin(\varphi) - \sin(\psi) * \cos(\varphi) & \cos(\psi) * \sin(\vartheta) * \cos(\varphi) + \sin(\psi) * \sin(\varphi) \\ \sin(\psi) * \cos(\vartheta) & \sin(\psi) * \sin(\vartheta) * \sin(\varphi) + \cos(\psi) * \cos(\varphi) & \sin(\psi) * \sin(\vartheta) * \cos(\varphi) - \cos(\psi) * \sin(\varphi) \\ -\sin(\vartheta) & \cos(\vartheta) * \sin(\varphi) & \cos(\vartheta) * \cos(\varphi) \end{bmatrix}$$

The orientation matrices can then be multiplied by a correction rotation matrix. This moves the orientation from the Earth frame to the camera frame.

$$R_{O,C} = R_C * R_O$$

Finally, the new orientation matrix can then be converted back to Euler angles.

$$\varphi_C = \tan^{-1}\left(\frac{R_{O,C}^{3,2}}{R_{O,C}^{3,3}}\right)$$

$$\vartheta_C = -\sin^{-1}(R_{O,C}^{3,0})$$

$$\psi_C = \tan^{-1}\left(\frac{R_{O,C}^{2,1}}{R_{O,C}^{1,1}}\right)$$

This results in orientation measured by the gloves in the camera frame and can be directly compared to the orientation measured by the fusionTrack500. Note that in practice the arctan2 function is used instead of arctan. Having found the rotation matrix of the glove with respect to the Earth frame, a correction rotation matrix R_C is necessary to move the rotation matrix from the Earth frame to the fusionTrack 500 reference frame. Since the relative orientation between the Earth frame and the camera frame is unknown, the correction rotation matrix must be found using data collected when the gloves are stationary.

By collecting the stationary data over approximately 60 seconds at the beginning of each dataset, during which the gloves do not move, it is possible to find the correction matrix. Since the glove is stationary and, therefore, the real orientation remains constant, it can be assumed that the error on the mean values is zero. This allows us to assume that the mean of the values measured by the gloves is equal to the mean values output by the fusionTrack 500 if both were in the same reference frame. This is used to numerically find a correction orientation matrix that aligns the values output by the gloves and by the fusionTrack 500. This is achieved in Python with the following bit of code.

```
error_min = 10000#Defines starting error between glove and fusionTrack data
error_threshold = 0.005#Defines how small the error must be in the end, in degrees
a_min = 0#angle about X that results in minimal error
b_min = 0#angle about Y that results in minimal error
c_min = 0#angle about Z that results in minimal error

a_start = 0.0
a_end = 2*np.pi
a_step = np.pi/2
b_start = 0
b_end = 2*np.pi
b_step = np.pi/2
c_start = 0
c_end = 2*np.pi
c_step = np.pi/2
#FIND correction matrix that moves from earth fram to camera frame
if findCM:
    while error_min > error_threshold:
        for a in np.arange(a_start,a_end,a_step):
            for b in np.arange(b_start,b_end,b_step):
                for c in np.arange(c_start,c_end,c_step):
                    R_X = np.array([[1,0,0],
                                    [0,np.cos(a),-np.sin(a)],
                                    [0,np.sin(a),np.cos(a)]])
                    R_Y = np.array([[np.cos(b),0,np.sin(b)],
                                    [0,1,0],
                                    [-np.sin(b),0,np.cos(b)]])
                    R_Z = np.array([[np.cos(c),-np.sin(c),0],
                                    [np.sin(c),np.cos(c),0],
                                    [0,0,1]])
                    correctionMatrix = np.matmul(np.matmul(R_Z,R_Y),R_X)
                    for i in range(len(senseOM)):#senseOM is a list of orientation matrices measured with the glove
                        temp = np.matmul(correctionMatrix,senseOM[i])
                        sense_roll[i] = np.arctan2(temp[2,1],temp[2,2])*180/np.pi
                        sense_pitch[i] = -np.arcsin(temp[2,0])*180/np.pi
                        sense_yaw[i] = np.arctan2(temp[1,0],temp[0,0])*180/np.pi
                    #compute the magnitude of the error
                    error = ((np.mean(sense_roll)-np.mean(VT_roll))**2+(np.mean(sense_pitch)-np.mean(VT_pitch))**2+(np.mean(sense_yaw)-np.mean(VT_yaw))**2)**0.5
                    if error < error_min:#save new minimum
                        print("a: ",a,"b: ",b,"c: ",c)
                        print("Error: ",error,"deg")
                        error_min = error
                        a_min = a
                        b_min = b
                        c_min = c
        #define new range of angles and angle step
        a_start = a_min-a_step
        a_end = a_min+a_step
        a_step = a_step/2
        b_start = b_min-b_step
        b_end = b_min+b_step
        b_step = b_step/2
        c_start = c_min-c_step
        c_end = c_min+c_step
        c_step = c_step/2

    print("Min a: ",a_min*180.0/np.pi,"Min b: ",b_min*180.0/np.pi,"Min c: ",c_min*180.0/np.pi,"Error: ",error_min)
```

Figure 12: Optimization Algorithm for Alignment Error Reduction in Sensor Data Fusion

This algorithm finds the optimal correction matrix by correcting the data from the gloves with all possible correction angle combinations and looking for the angle combination at which the error between the corrected glove data and fusionTrack 500 data is minimal. This error is defined as follows, where the subscript C represents the glove data multiplied with the current correction matrix and the subscript FT represents data obtained from the fusionTrack 500.

$$error = \sqrt{(\overline{\varphi_C} - \overline{\varphi_{FT}})^2 + (\overline{\vartheta_C} - \overline{\vartheta_{FT}})^2 + (\overline{\psi_C} - \overline{\psi_{FT}})^2}$$

However, if this calculation were performed for all possible combinations, it could take days or even years, depending on the size of the discretization step. Instead, the initial discretization step is set to be large, significantly reducing the number of combinations. At the end of each cycle, the range of angles is narrowed based on the previous discretization step, and the new discretization step is halved.

This approach continually refines the correction matrix, progressively reducing the error until it falls below a predefined threshold. For this experiment, the error threshold was set at 0.0001 degrees. Once the optimal correction matrix is determined, it can be applied to all collected data points, allowing for direct comparison between the Euler angles from the gloves and the fusionTrack 500.

5.2.2 Example of Orientation Processing

This section presents the intermediate results of the previously described data processing method. It is important to note that the 'Mean Absolute Error' shown in the figures is not relevant when determining the correction matrix since it is absolute; instead, it is necessary when comparing the accuracy of different datasets in the following sections. The raw data captured experimentally using either the Nova SenseGlove or the glove developed in this thesis provides a time series of the Euler angles of the hand orientation. An example of post-processing of the raw data is demonstrated in figures Figure 13, Figure 14, Figure 15, and Figure 16.

Figure 13 displays raw time-series data of the Euler angles of the data glove compared to the ones of the visual tracker. It can clearly be seen that there is a large difference in values, which can be attributed to the different reference frames used. To find the correction matrix for this dataset, the static parts of the data set corresponding to approximately the first minute of the time-series need to be isolated. The isolated measurements can be visualized in Figure 14.

With the static part isolated, it is possible to run these datapoints through the correction matrix finding algorithm. The correction matrix can be applied to the static part of the data set, resulting in the values showcased in Figure 15. These data points clearly show little mean offset, meaning that the correction algorithm successfully translated the reference frame of the data glove onto the reference frame of the fusionTrack500.

Finally, the ideal correction matrix is saved and applied to the whole dataset, which can then be trimmed to include only the moving part, and the mean absolute error can be computed. The result of this is shown in figure 16. At the same time, the mean error is computed at each time point and visualized with respect to time. The results of this operation can be visualized in Figure 17, clearly showing that the time series data of the data glove effectively follows the movement recorded by the visual tracker.

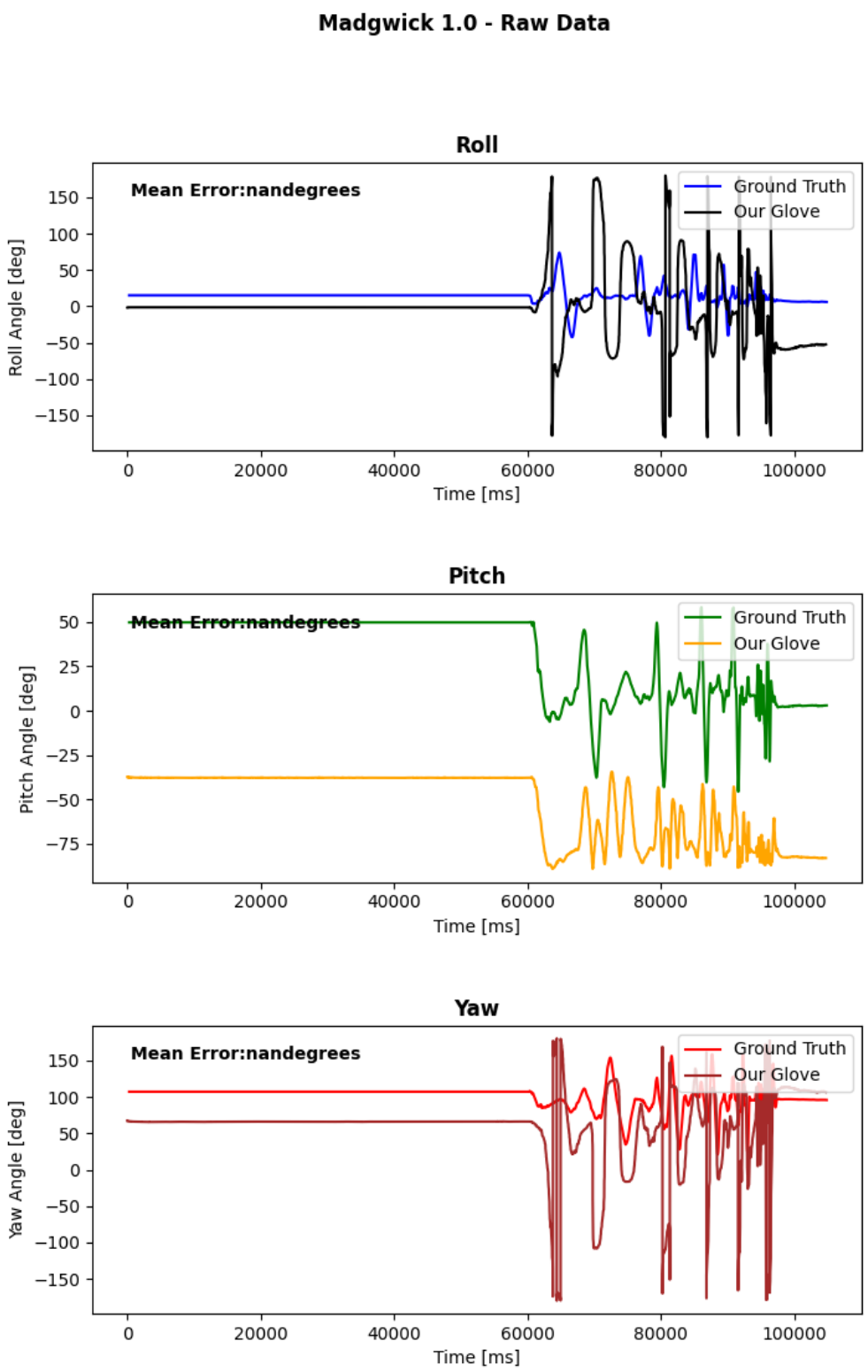

Figure 13: Example Raw Data of Roll, Pitch and Yaw from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1

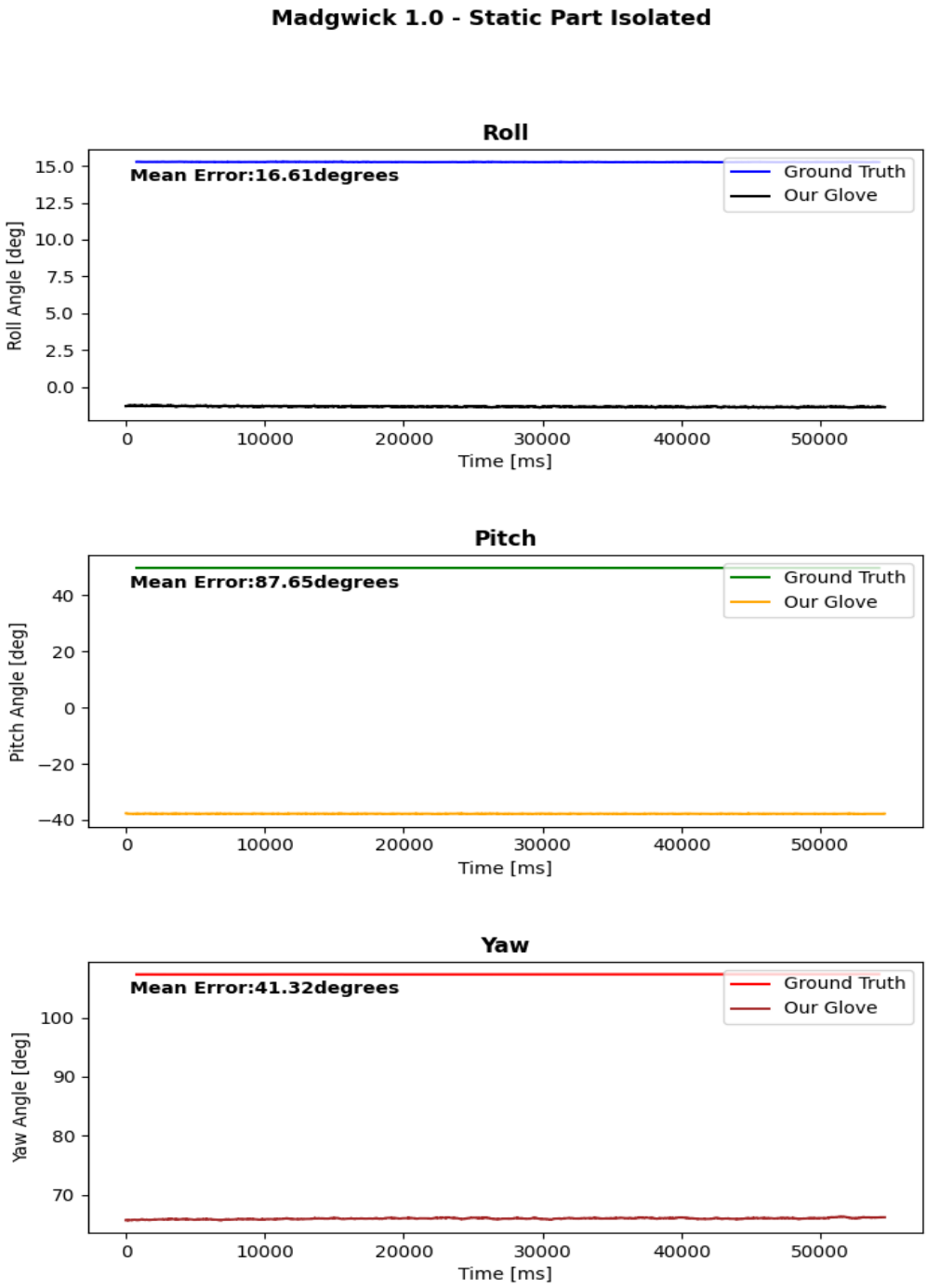

Figure 14: Example Static Data of Roll, Pitch and Yaw from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1

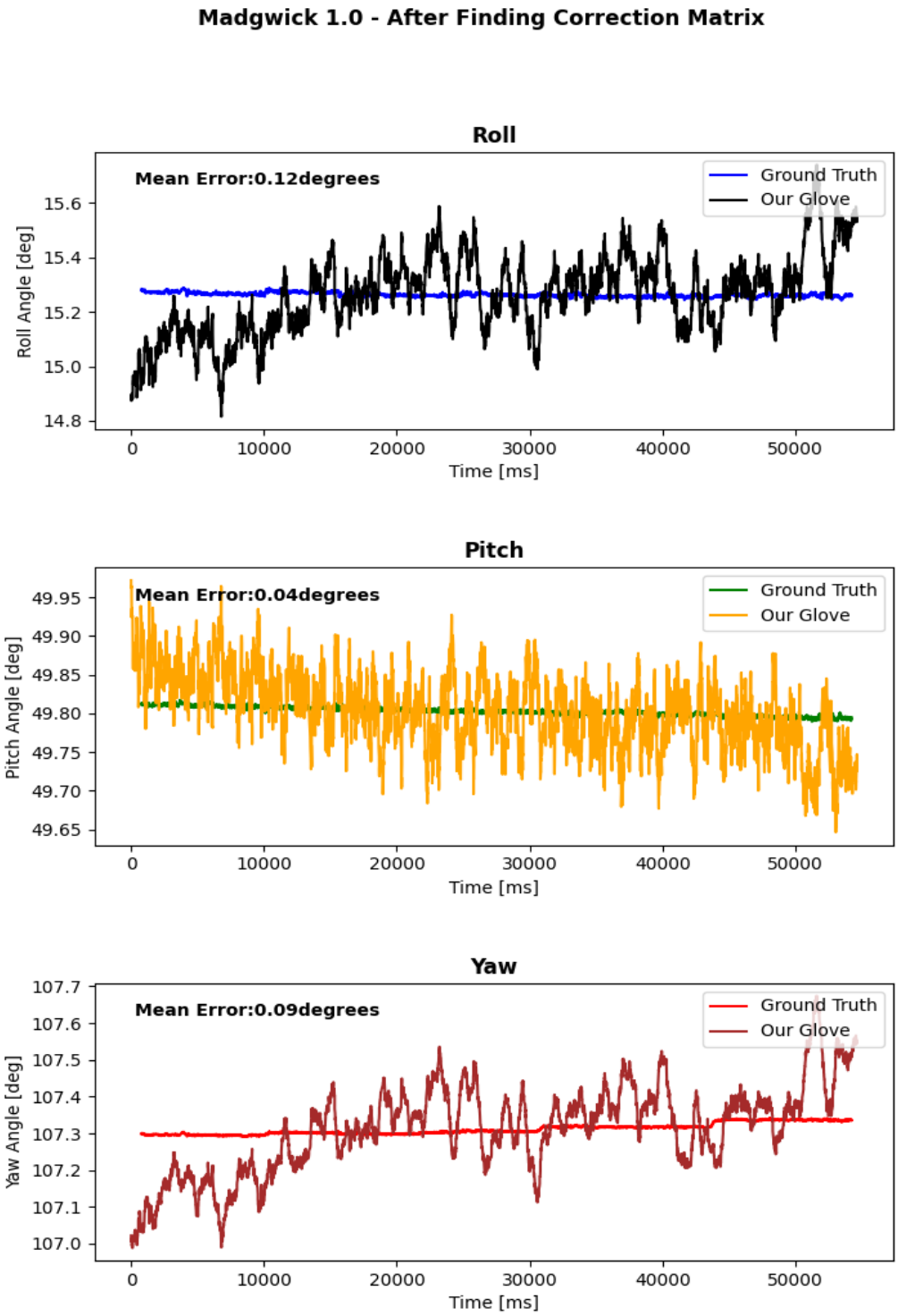

Figure 15: Example Static Data of Roll, Pitch and Yaw, Corrected by Correction Matrix, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1

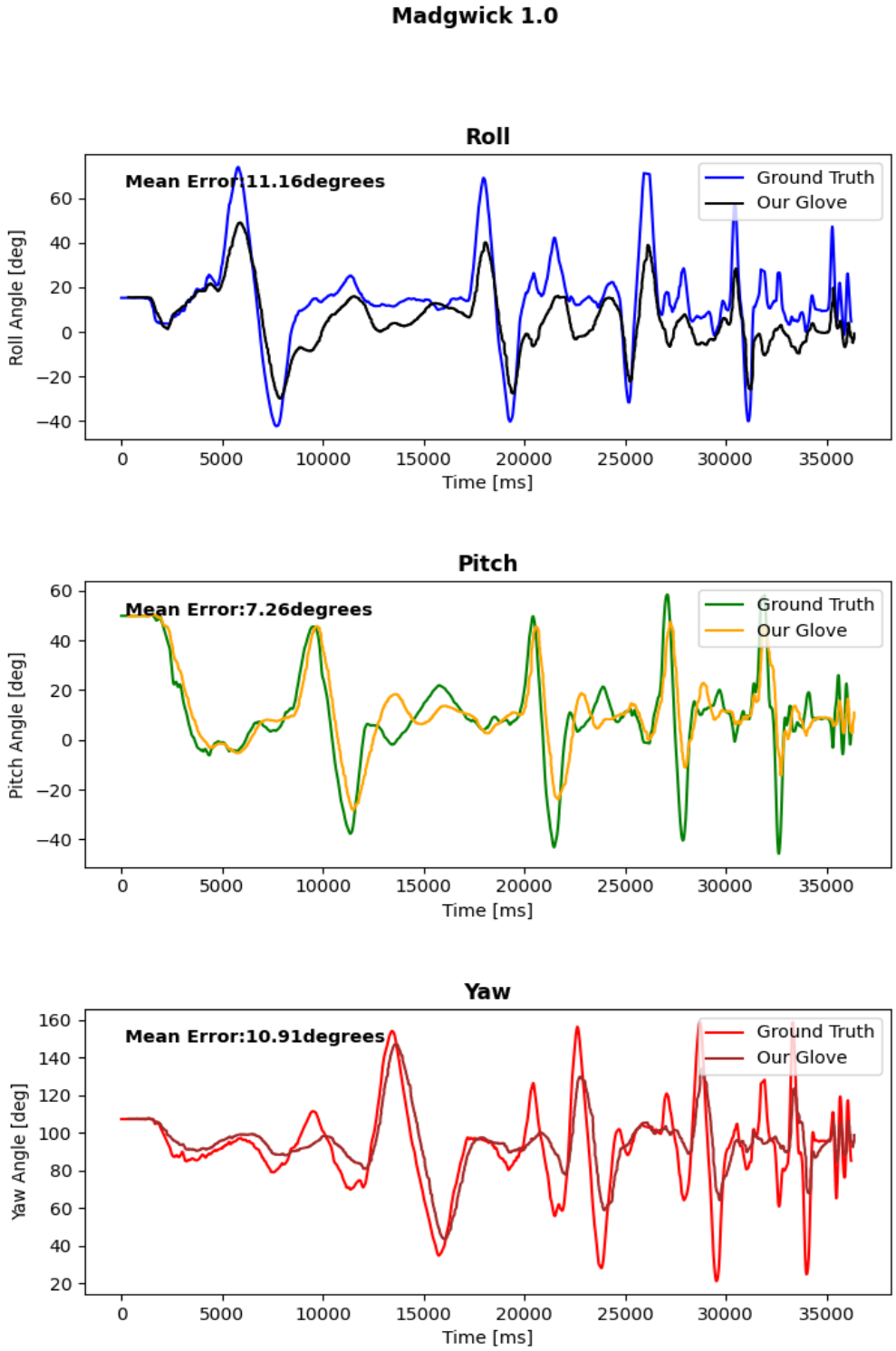

Figure 16: Example Processed Data of Roll, Pitch and Yaw from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1

5.2.3 Results: Orientation Tracking

The following set of figures shows the processed data of all the different collected datasets. These correspond firstly to the processed data of the HSDK data glove using the Madgwick sensor fusion algorithm with values for beta, namely the gradient factor, equal to 1.0, 1.25, 1.5, 1.75, and 2.0. These can be found in Figure 17, Figure 18, Figure 19, Figure 20, and Figure 21. In addition to these first five measurements, the HSDK glove is also evaluated using an Extended Kalman Filter (EKF), as shown in Figure 22. Lastly, in Figure 23, the state-of-the-art SenseGlove Data Glove is evaluated according to the same methodology, allowing for a comparison between the tracking for each data set. Additionally, the results of the absolute mean errors are summarized in Table 10.

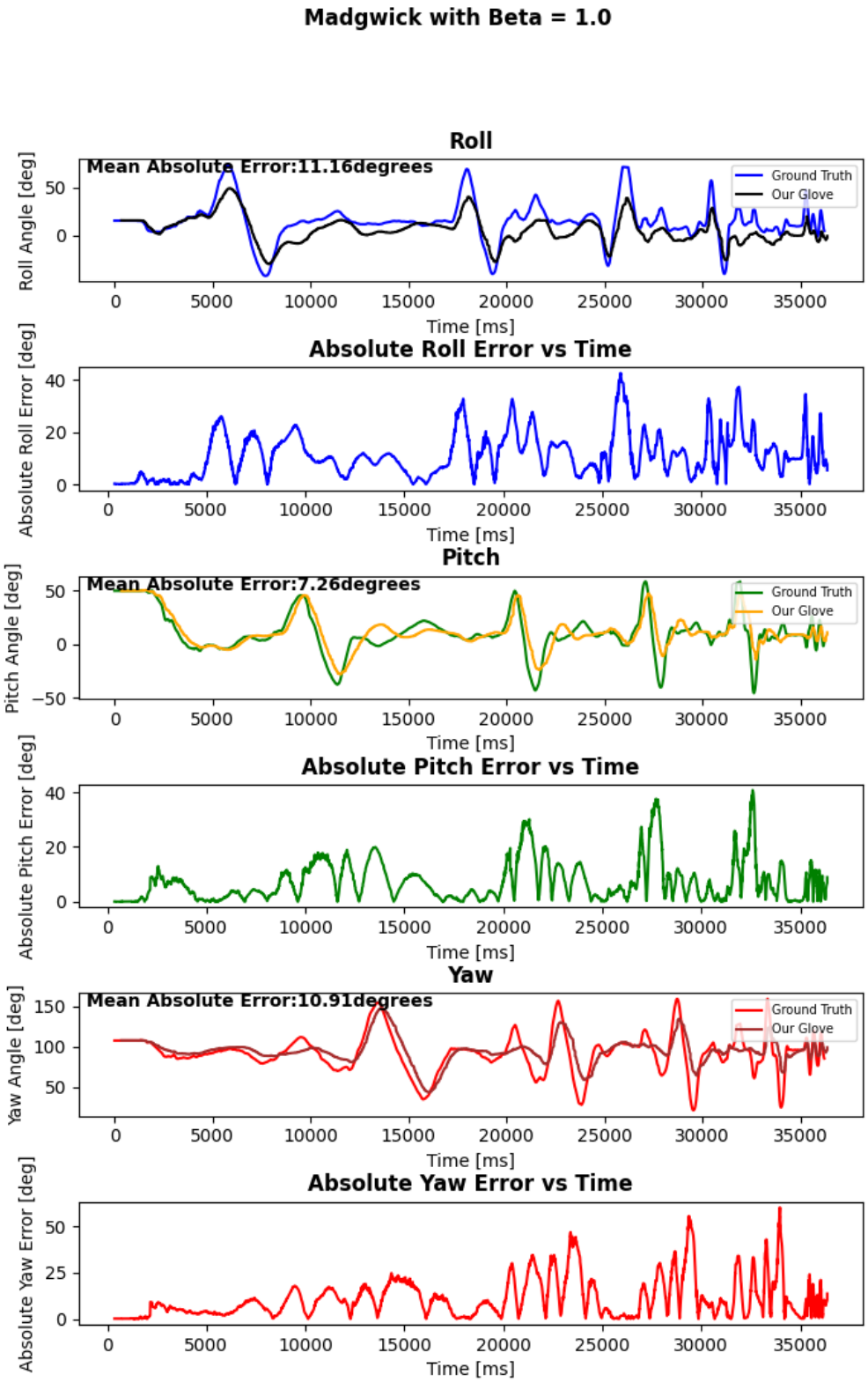

Figure 17: Processed Data and Error Timeseries of Roll, Pitch and Yaw, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1

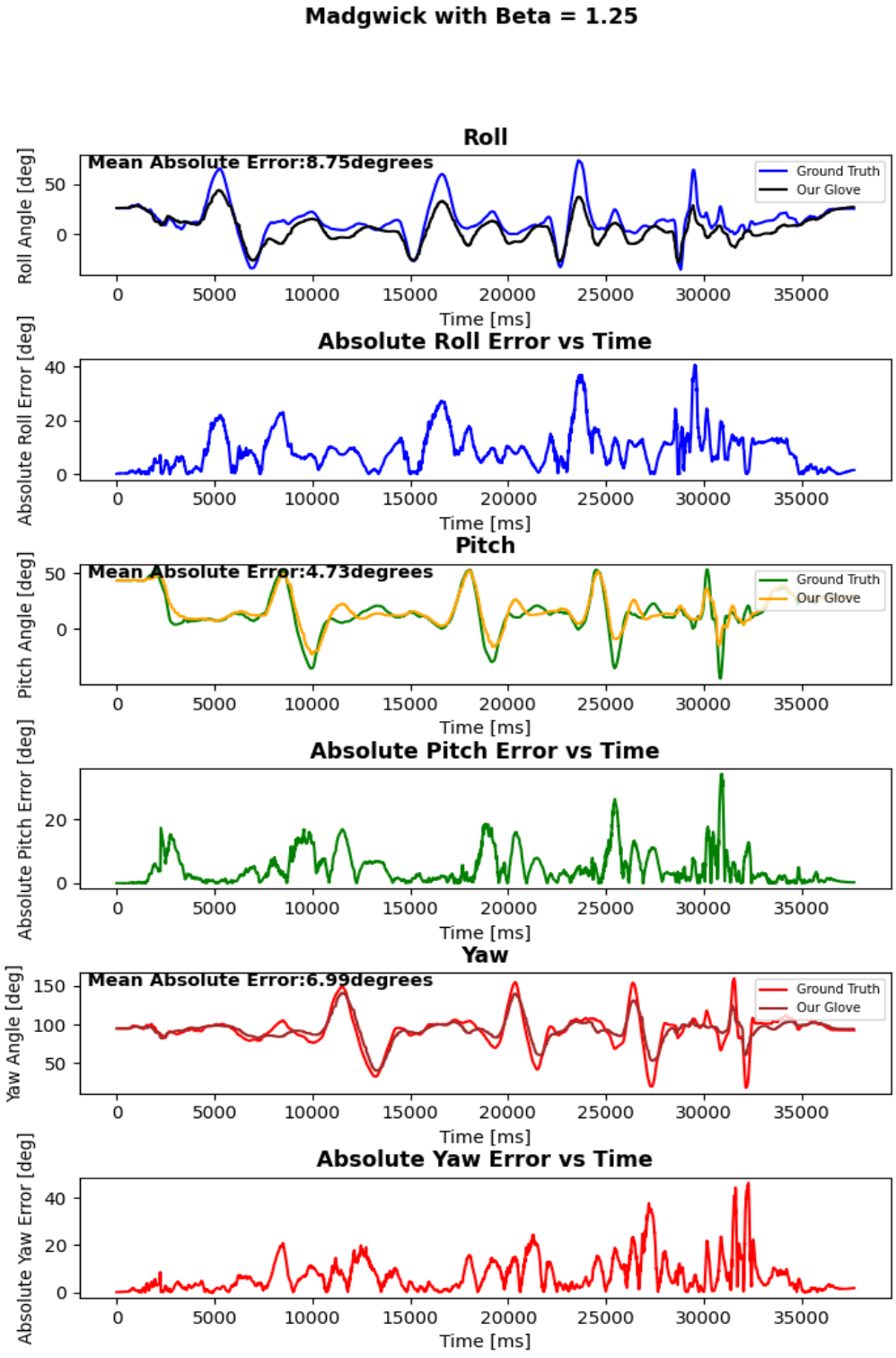

Figure 18: Processed Data and Error Timeseries of Roll, Pitch and Yaw, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1.25

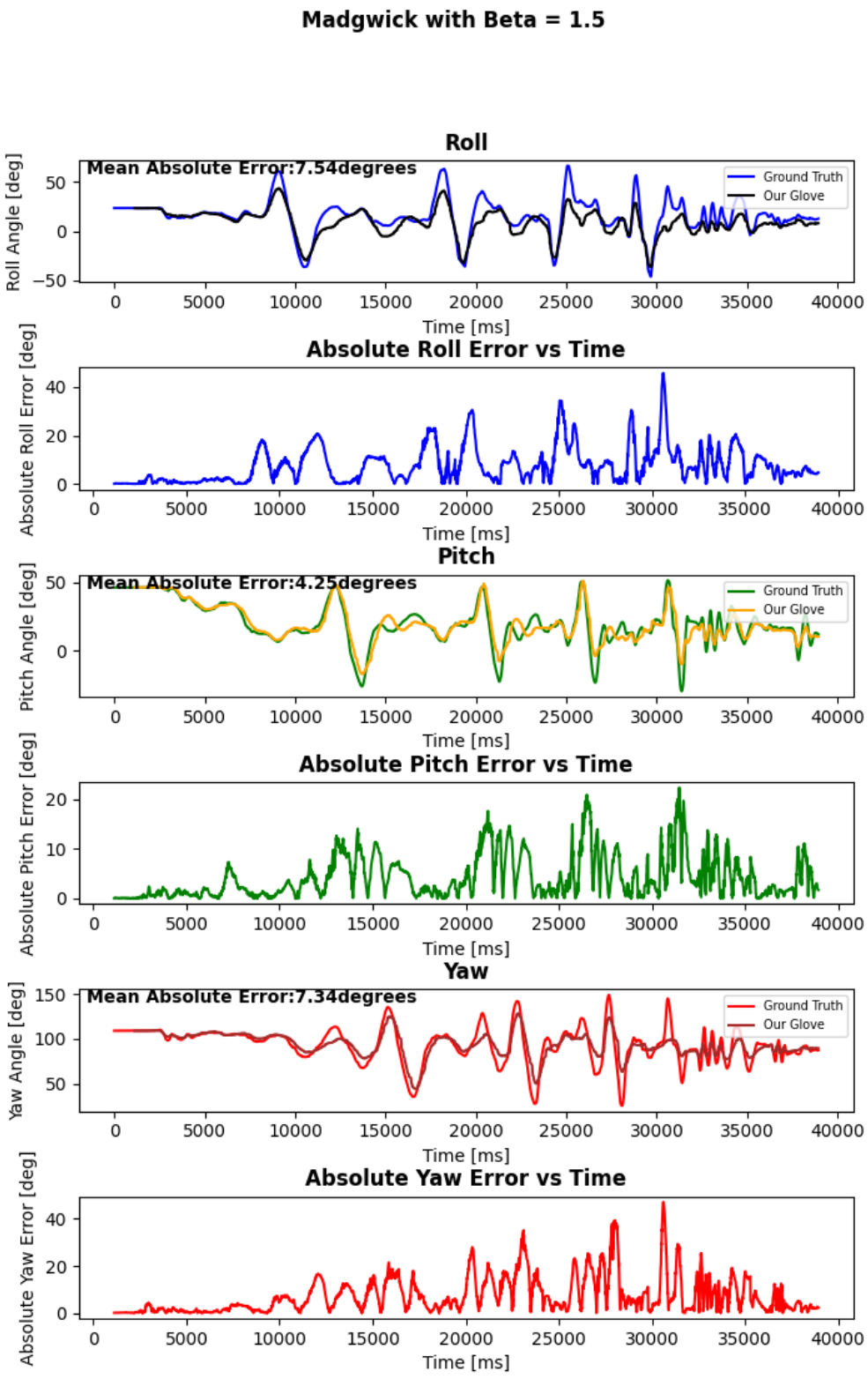

Figure 19: Processed Data and Error Timeseries of Roll, Pitch and Yaw, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1.5

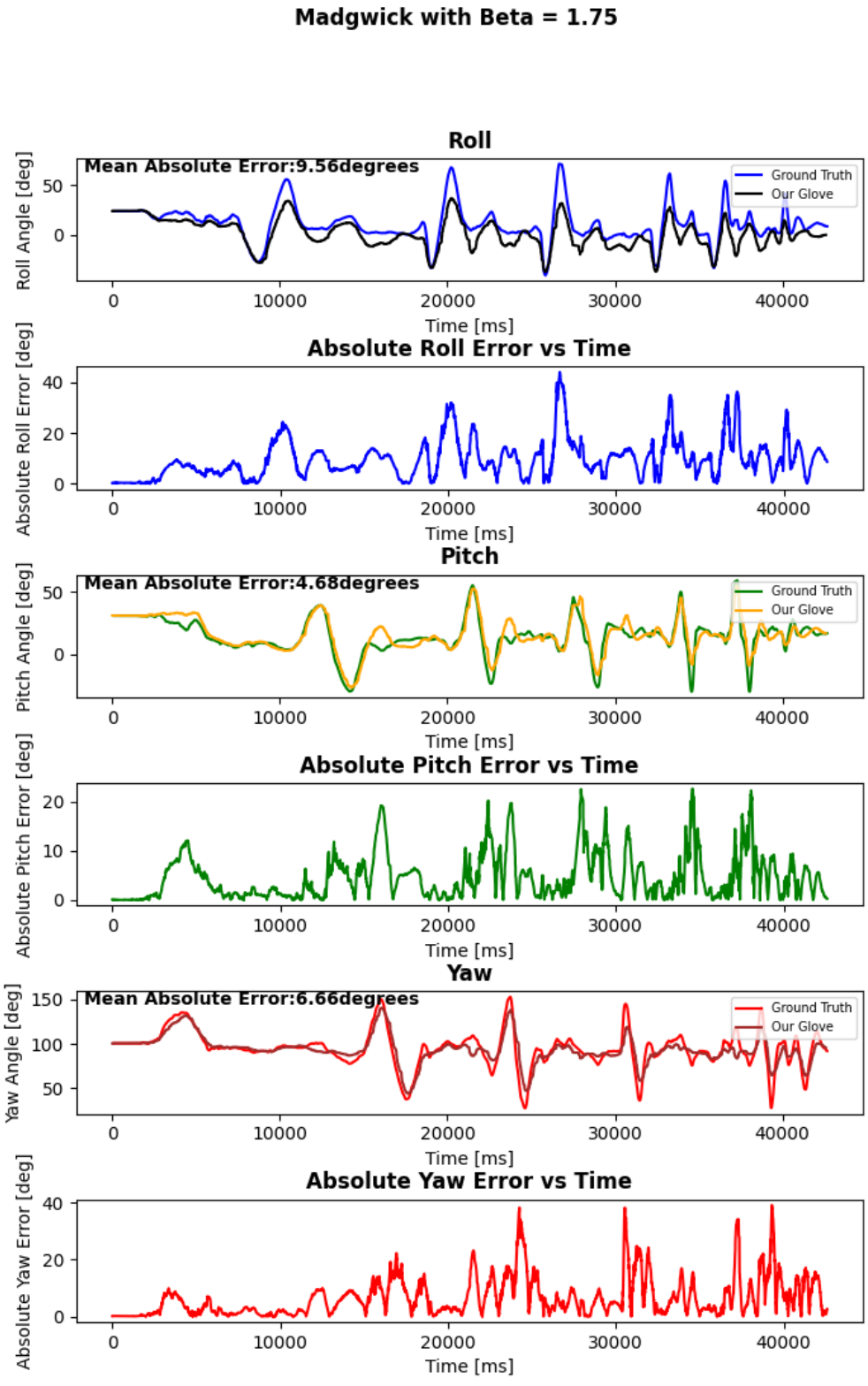

Figure 20: Processed Data and Error Timeseries of Roll, Pitch and Yaw, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta=1.75

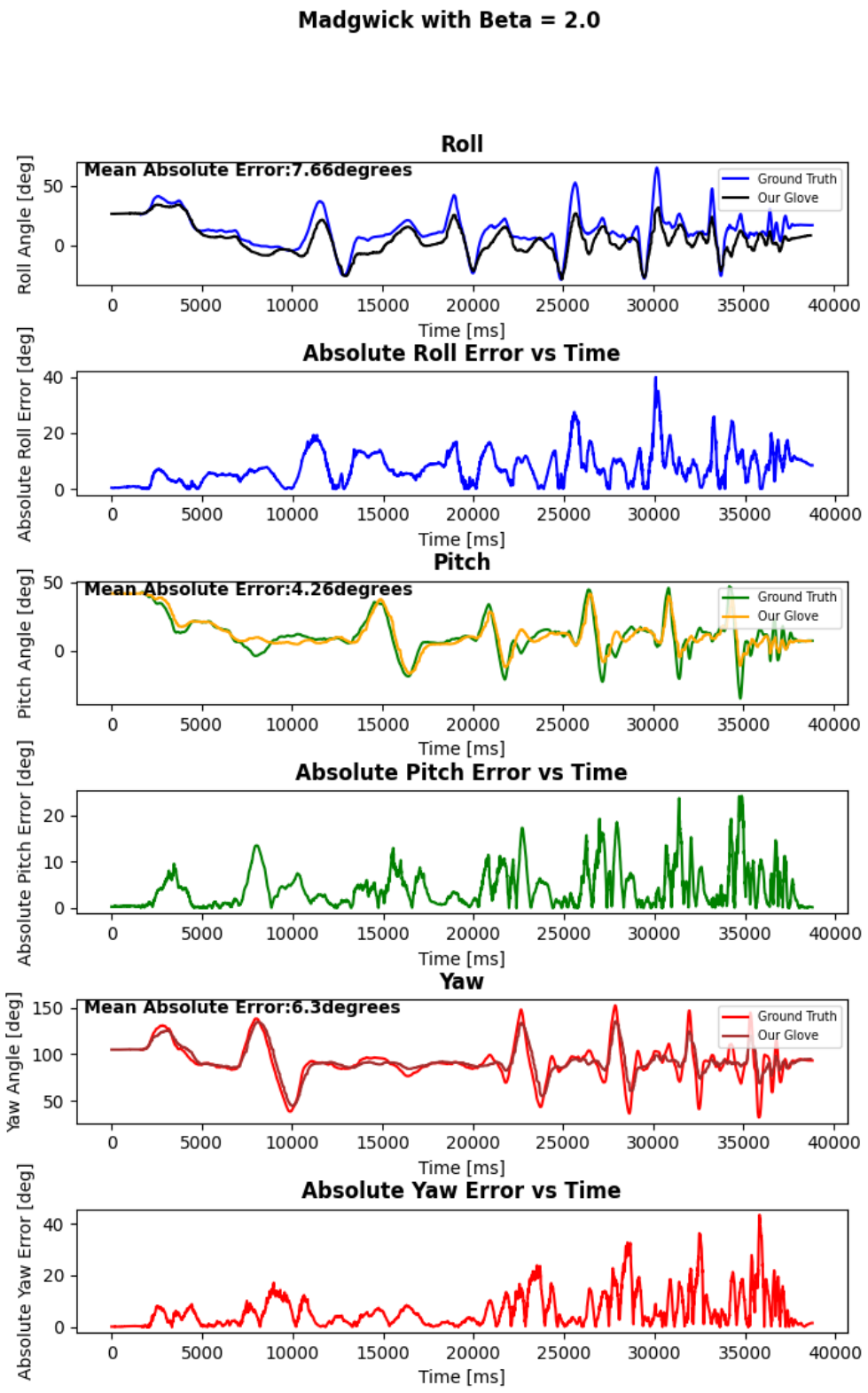

Figure 21: Processed Data and Error Timeseries of Roll, Pitch and Yaw, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Madgwick filter and gradient beta= 2

EKF

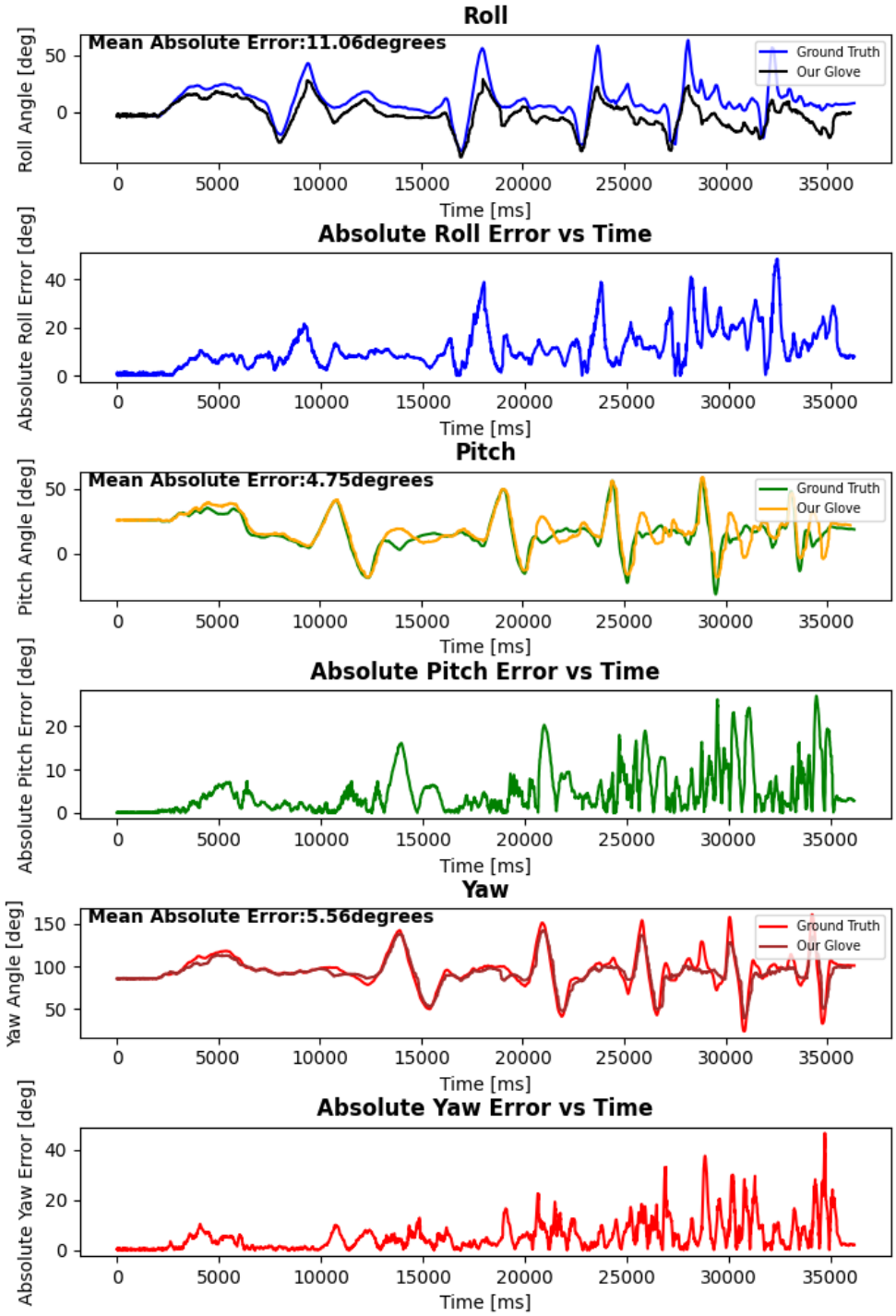

Figure 22: Processed Data and Error Timeseries of Roll, Pitch and Yaw, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Extended Kalman Filter (EKF)

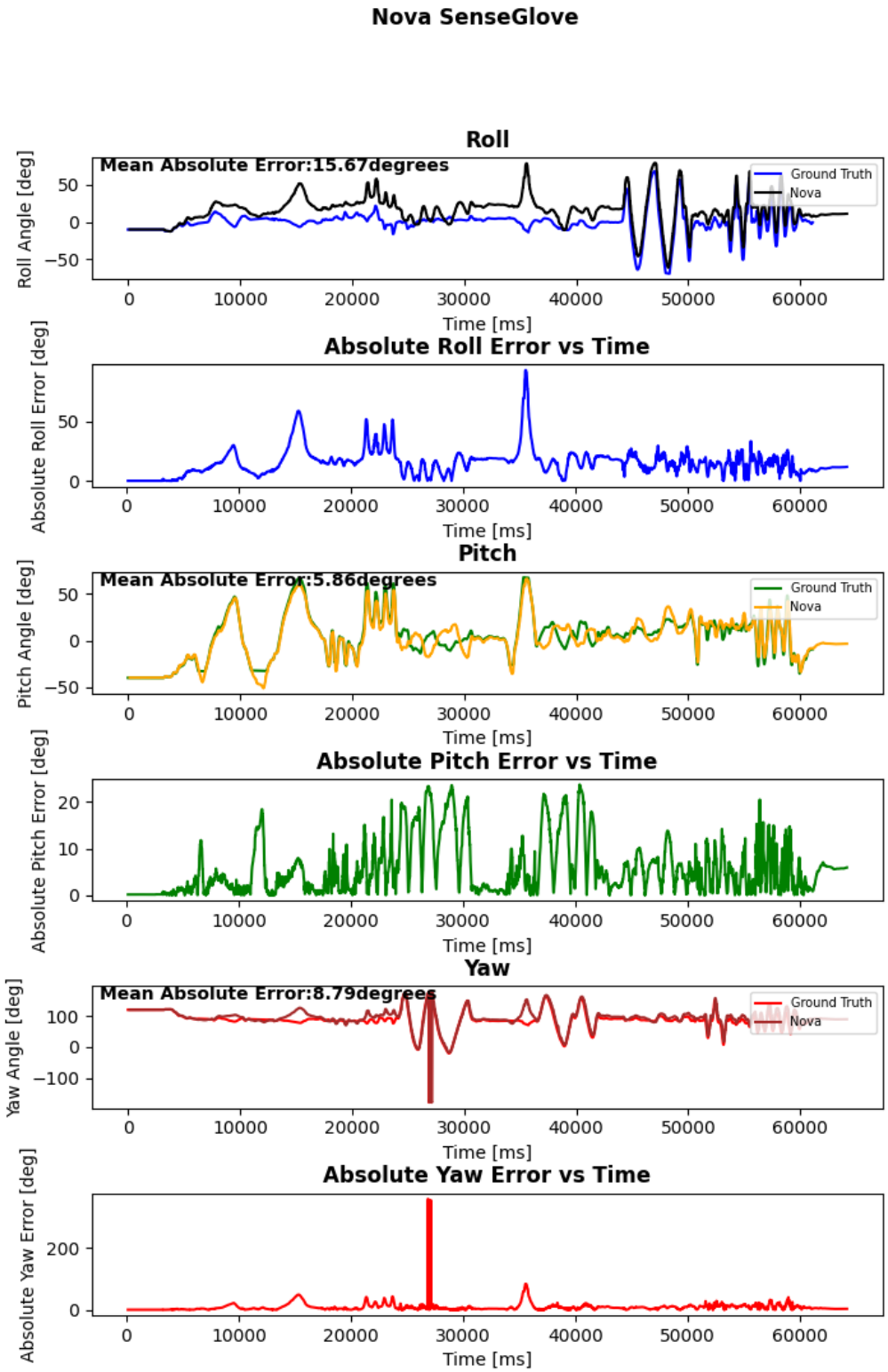

Figure 23: Processed Data and Error Timeseries of Roll, Pitch and Yaw, from SenseGlove Nova Data Glove and Visual Tracker (Ground Truth)

Table 10: Comparison of Roll, Pitch, and Yaw Absolute Mean Errors Across Different Sensor Fusion Algorithms and Data Gloves

	Roll Absolute Mean Error [degrees]	*Pitch Absolute Mean Error [degrees]*	*Yaw Absolute Mean Error [degrees]*
Madgwick 1.0	11.16	7.26	10.91
Madgwick 1.25	8.75	4.73	6.99
Madgwick 1.5	7.54	4.25	7.34
Madgwick 1.75	9.56	4.68	6.66
Madgwick 2.0	7.66	4.26	6.3
EKF	11.06	4.75	5.56
Nova SenseGlove	15.76	5.86	8.79

5.2.4 Processing the Finger Tracking

When processing the finger tracking data, only the yaw output by the fusionTrack 500 needs to be considered as the finger only bends about one axis. At the same time, the fusionTrack500 can be used to scale the glove data to correct angle magnitudes. Since the gloves output an arbitrary value, it needs to be scaled to relative angle changes. This can be done by simply looking at what maximum and minimum angle values correspond to maximum and minimum values output by the gloves and linearly scaling between them.

Given the time series of the index finger flexions provided in Figures 24, 25, and 26, it is possible to scale the raw index finger data from the HSDK glove and compare it to the angle measured by the fusionTrack 500. The scaling formula used for this post-processing step for both the HSDK glove and the SenseGlove Nova is the following:

$$glove_{corrected} = \frac{(glove_{raw} - glove_{min})}{(glove_{max} - glove_{min})} * (angle_{min} - angle_{max}) + angle_{max}$$

Where, $glove_{raw}$ is the time series data of the raw data set with limits $glove_{max}$ and $glove_{min}$ while $angle_{min}$ and angle and $angle_{max}$ are the range within which the time series is scaled. The following section provides an overview of this normalization step, showcasing how the raw data is scaled to allow comparison with the data from the fusionTrack visual tracker.

5.2.5 Example of Finger Tracking Processing

Following the discussion on the post-processing steps applied to the finger tracking data, Figure 18 and Figure 19 provide a depiction of how the scaling allows comparison between the measurements. In Figure 24, the measurements of the index finger flexions do not align between data sets. After applying the post-processing, Figure 25 shows that the post-processed data is much more in line with expectations, allowing a fair comparison between Ground Truth and the measurement taken from the HSKD glove. The same procedure is then replicated using the raw measurements of the flexions from the SenseGlove Nova data glove, showcasing the results in Figure 26.

5.2.6 Results: Orientation Tracking

The following figures, Figure 24, Figure 25, Figure 26, showcase the example Raw Data of the HSDK glove, the processed data of the index flexions, and the processed data from the SenseGlove Nova index flexion, all with respect to the visual tracker acting as the ground truth.

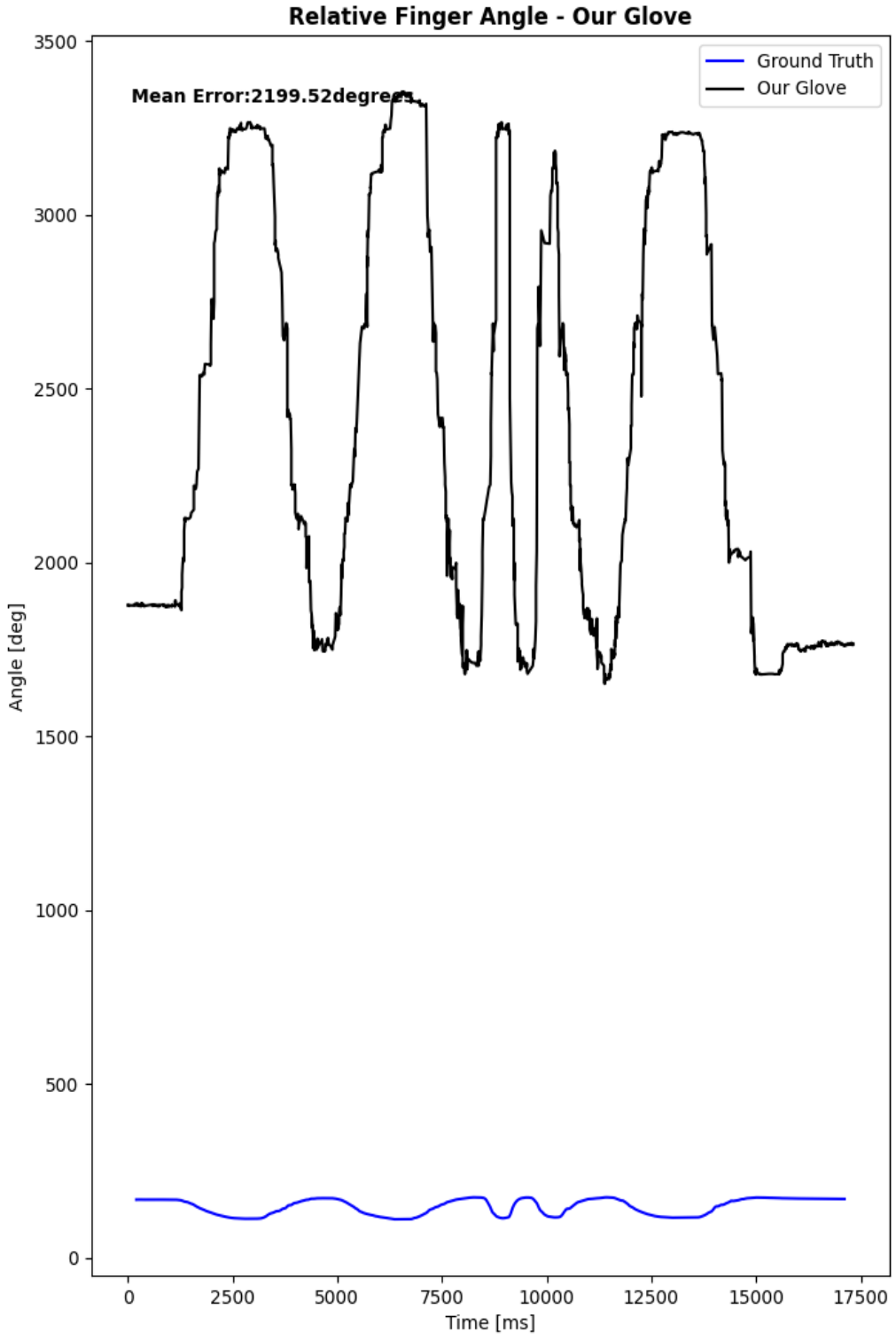

Figure 24: Example Raw Data Timeseries of Finger Flexions, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth)

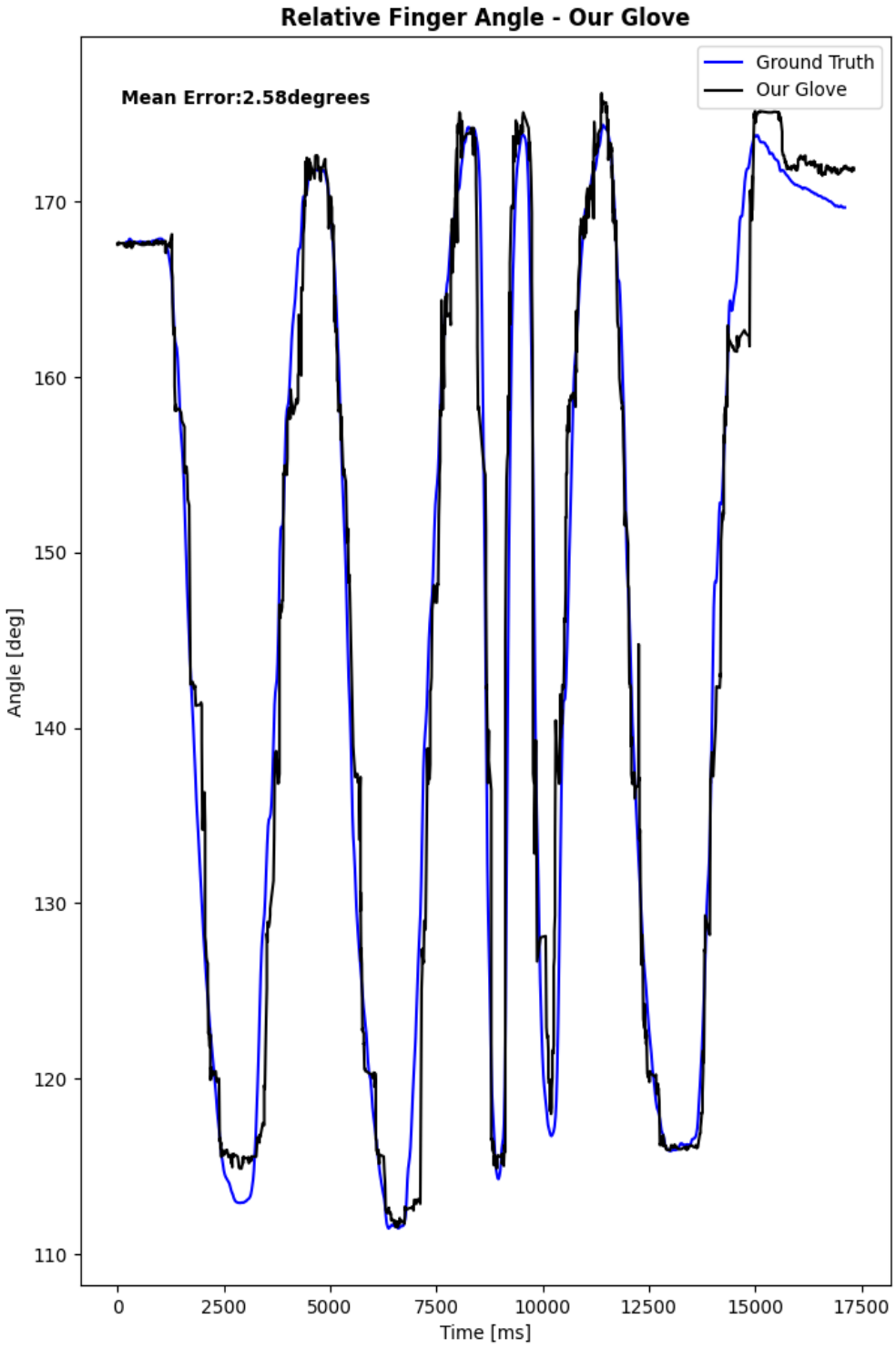

Figure 25: Processed Timeseries Data of Finger Flexions, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth)

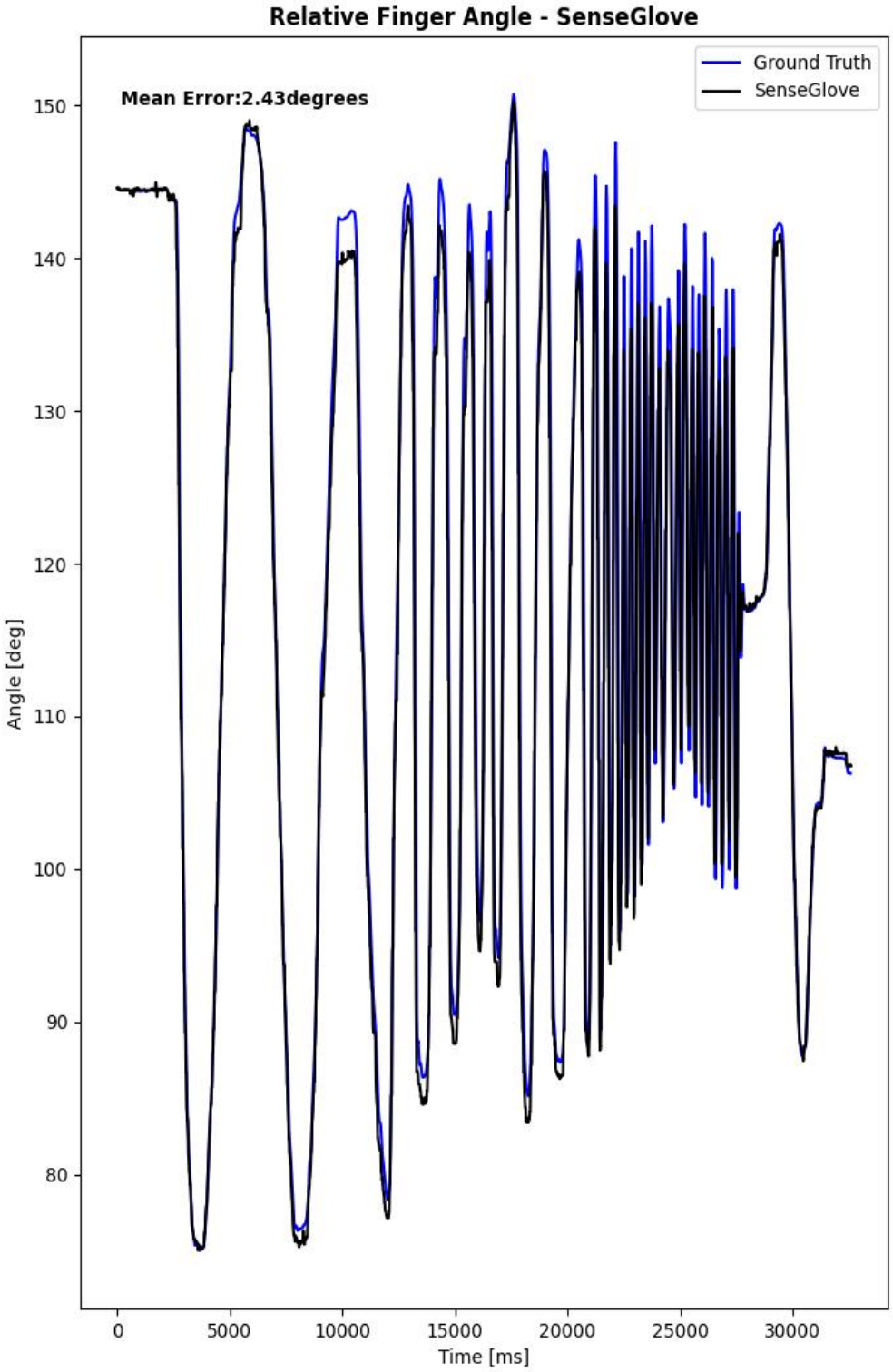

Figure 26: Processed Timeseries Data of Finger Flexions, from the SenseGlove Nova Data Glove and Visual Tracker (Ground Truth)

5.3 Experimental Data Discussion

5.3.1 Orientation Precision and Drift

When one looks at Figure 27, which shows the static part of the corrected data, the precision of the HSDK glove can be clearly seen. The constantly changing values are the result of any noise or slight disturbances present on the sensors, as well as any numerical errors. The same figure can be plotted for the EKF and the Nova SenseGlove.

From these three figures, Figure 15, 27, and Figure 28, one can see that the Nova SenseGlove has the highest precision. The EKF, on the other hand, can be found to have the lowest precision.

Another aspect that can be observed is any potential drift. The Madgwick filter seems to experience small drift, about 0.2 to 0.5 degrees per minute, depending on the axis. However, it is difficult to conclude whether it would continue to drift over a longer period. It is possible that the slightly changing trend is due to the convergence of the gradient descent algorithm.

To further analyze this, future experiments would need to be conducted to see whether the algorithm truly drifts and, if so, what the cause of the drift would be.

The EKF, on the other hand, does not seem to experience any drift. Although there is a slight slope to the data, it can be seen that this slope is also output by the ground truth; hence, the glove must have been slightly shifting during this portion of data collection.

The Nova SenseGlove on the other hand, does not experience any drift at all, quite the opposite. While the ground truth is slightly shifting, the output of the Nova SenseGlove remains perfectly constant. While not much information could be found about the sensor fusion algorithm implemented on the Nova SenseGlove, it is clear the glove does not try to estimate its orientation at all times. Rather, when it detects angular rates below a certain value, it simply continues returning the same value. It can only be concluded that this is done on purpose to achieve very stable and artificially precise values when the glove is at an apparent standstill.

EKF

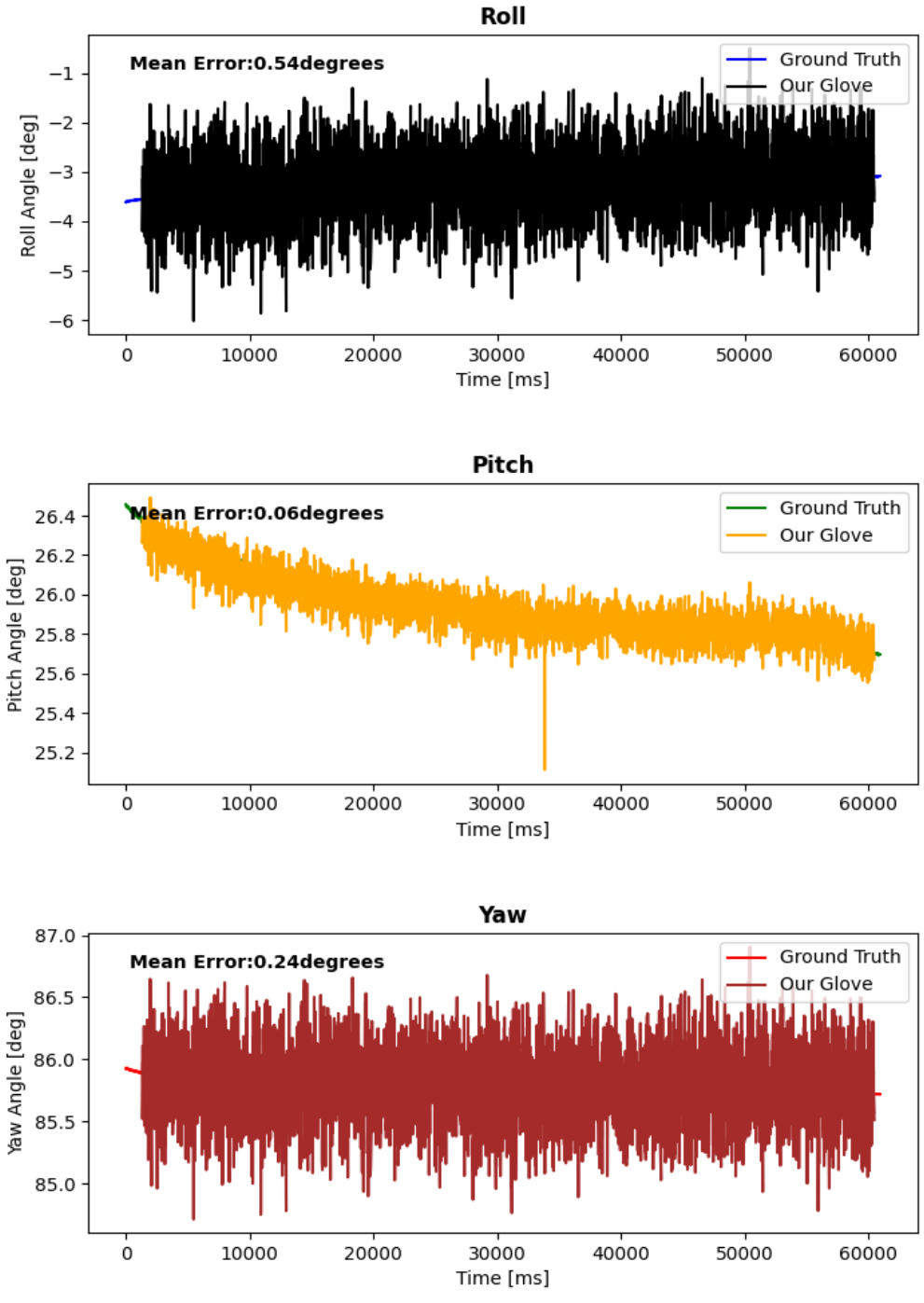

Figure 27: Static Processed Timeseries Data of Roll, Pitch and Yaw, Corrected by Correction Matrix, from the HSDK (Our Glove) Data Glove and Visual Tracker (Ground Truth), Using Extended Kalman filter

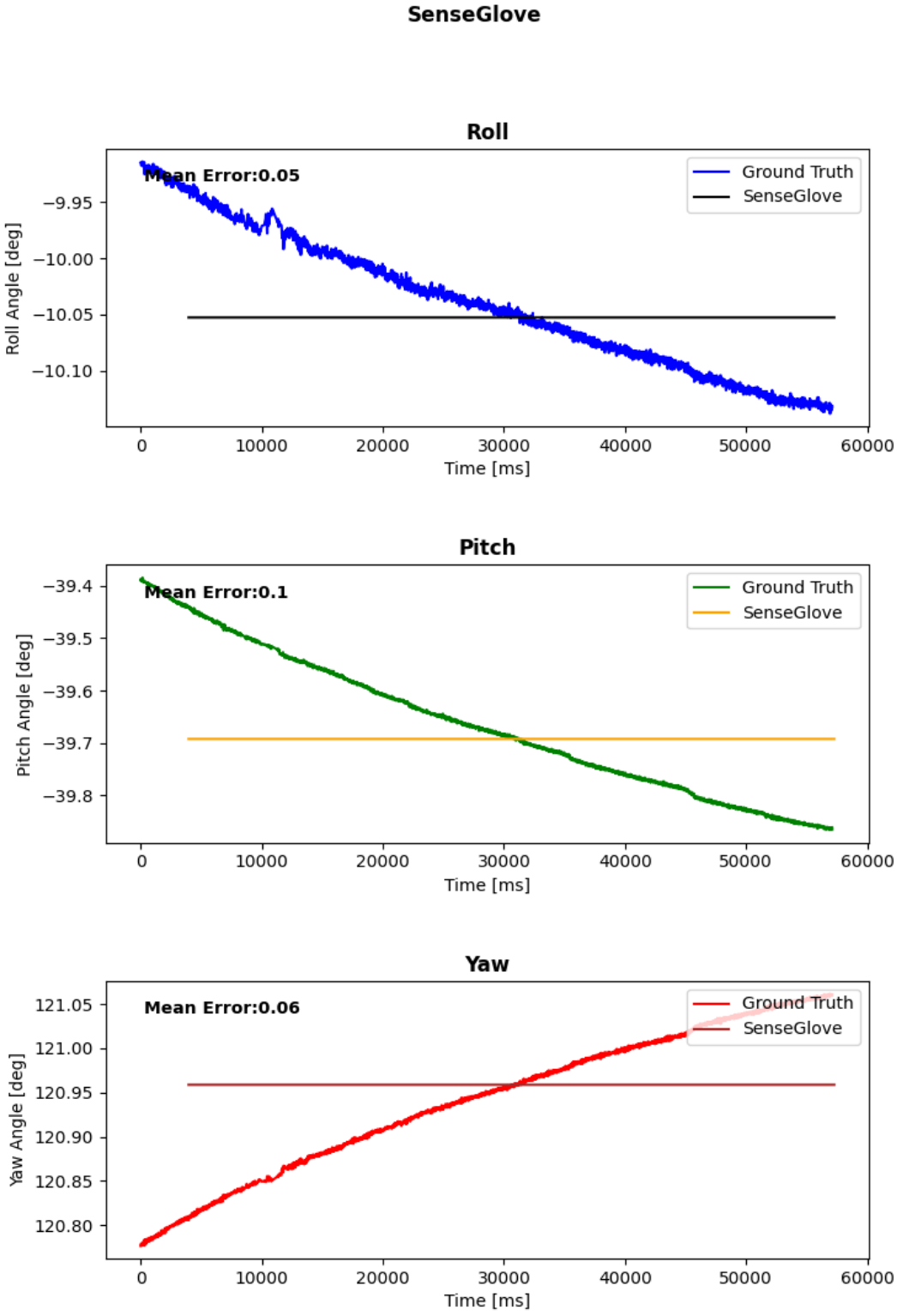

Figure 28: Static Processed Timeseries Data of Roll, Pitch and Yaw, Corrected by Correction Matrix, from the SenseGlove Nova Data Glove and Visual Tracker (Ground Truth)

5.3.2 Orientation Accuracy

Table 10 summarizes the accuracy results, and the data shows that the HSDK glove is capable of achieving higher mean absolute accuracy than the sense glove. While it is satisfactory to see that the performance is comparable, the data needs to be analyzed a bit further.

When one looks at the results figures closer, there is a clear offset between the roll values of the gloves and the ground truth. While the cause of the offset is unknown at the time of writing this paper, it must be considered when drawing conclusions. This offset clearly enlarges the mean absolute error; as such, the computed mean absolute errors cannot be taken as absolute truth. However, the offset is also present in every single dataset, which is only logical since the data collection and processing method was consistent between datasets. What cannot be determined is whether this offset is the same for each dataset or differs. As such, the mean absolute errors can be compared, but one must remain skeptical about the results.

When looking at the differences in performance between Madgwick filters with different beta values, a decreasing trend can be seen with increasing beta values. This is because a higher beta value makes the gradient descent algorithm more aggressive and allows it to better track the orientation at higher angular rates.

EKF, on the other hand, has a higher roll mean absolute error than most of the Madgwick filters; however, the pitch and yaw results are the same or lower. Taking into consideration the lower precision, one could conclude that the Madgwick is a better choice overall. However, it must be remembered that the EKF also has nine parameters in total that can be tuned; $Q_{max}, Q_{slope}, Q_{min}, R_{acc,slope}, R_{acc,min}, R_{acc,max}, R_{mag,slope}, R_{mag,min}, R_{mag,max}$. As such, the results shown in this paper are only for one combination of these parameters. While the parameters were tuned to achieve a balance between drift, accuracy, and precision, they could be retuned to get different performances. By changing the relative uncertainties between the gyroscope, accelerometer, and magnetometer, high accuracy can be achieved at the cost of experiencing small drift or no drift at the cost of precision and accuracy.

Lastly, the performance of the HSDK glove can be compared to that of the Nova SenseGlove. From the data, it can be found that the accuracy achieved by the glove developed in this paper is higher than that of the Nova SenseGlove. Since no relevant information could be found on the algorithm used in the Nova SenseGlove, it cannot be determined why these results were obtained.

In conclusion, it can be safely said that the performance of the HSDK glove is similar to that of the Nova SenseGlove. However, due to the limitations of the experimental setup, no empirical conclusions can be made. This is acknowledged as a limitation, but due to the limited time within the context of this thesis, it cannot be addressed.

5.3.3 Finger Tracking Accuracy

Based on Figure 25 and Figure 26, it is clear the Nova SenseGlove has better finger-tracking resolution than the HSDK glove. At the same time, the accuracy is better. This is mainly due to the difference in mechanical construction. The Nova SenseGlove attaches to the hand with two straps and provides better support for the finger sensor. The glove developed in this paper, on the other hand, only uses one strap to attach the base of the finger sensors to the hand, so the sensors shift significantly due to the forces produced by the springs.

Ultimately, although the HSDK glove lacks finger tracking performance, the finger sensors still create valuable additional information to the hand orientation that can be used for practical applications.

5.4 Future Experiment Improvements

As mentioned in previous subsections, the experiment performed in this paper to compare the performance of the HSDK glove to the state of the art has flaws that do not allow it to draw empirical conclusions. The main reason for that is that gloves were moved by hand, and as such, the movements were not consistent between datasets. Due to this, a large improvement to the experiment would be to use a robotic arm that could create consistent, repetitive movements.

6 CONCLUSIONS

This paper documents the development process of an open-source data glove platform. While the development of data gloves is not new, the goal and procedure chosen in this paper allow it to fill the gaps in the data glove domain. Current state-of-the-art gloves are made proprietary in one way or another, which hinders the accessibility to data gloves and the development of possible applications with them. The approach conducted in this thesis fills this gap and gives other engineers the opportunity to utilize the tools developed for their own applications.

In this thesis an open-source platform was developed in an attempt to centralize data glove development. With the ultimate goal of providing software and hardware templates to easily kickstart both academic and commercial data glove development, a GitHub repository was created, including a Docker-based software stack, PCB templates, and micro-controller firmware. With flexibility, ease of use, and cost as requirements, the platform was designed to incorporate tools, such as ROS2 and the ESP32-based microcontroller, in an attempt to standardize and streamline the development and deployment of data gloves.

The hardware was developed in a free, online-based PCB software so that anybody could change the electronics design to fit their needs. At the same time, low-cost, popular, and highly documented components were selected for further ease of use. The microcontroller source code was developed from scratch and made available with explanations for other engineers to utilize as necessary. All the CAD files created to manufacture the small hardware necessary to complete the glove were also made public. Lastly, the achieved low cost of the hardware allows it to be manufactured and further improved by other engineers. Future developments could include improving the finger sensors or focusing on other aspects of the glove, such as energy efficiency.

The software was developed through a Docker-based, containerized environment, allowing easy development and deployment of both back- and front-end applications while maintaining a high degree of flexibility in the selection of communication protocols. While difficulties in implementing serial communication protocols were encountered due to limitations imposed by docker on serial communication, the issue was circumvented through USB/IP. Nonetheless, the developed architecture allows the effective management of the microROS communication channel, thanks to the back-end server, while being cross-platform compatible and highly flexible. Additionally, the platform's integration with ROS2 environments and microROS-based microcontrollers means that while initially intended for data gloves, the developed platform can be extended to a larger, more diverse set of devices such as robots, further extending the flexibility of the platform. Ultimately, the developed architecture positions itself to easily enable additional developments. These next steps include implementing bi-directional communication between the client and agent, conducting further research on different serial communication passthrough methods into Docker containers, and attempting to use the glove in robotic teleoperation.

While the experiment conducted to compare the performance of the developed data glove to the state-of-the-art had shortcomings, it still showed basic data that allows to conclude that the glove developed in this thesis is close to the level of state-of-the-art, which creates a solid foundation for further development.

Despite the tight time constraints, the progress achieved has been extensive, showcasing significant potential of the HSDK. The open-source publication of all resources will enable this data glove to become the next state-of-the-art data glove.

7 REFERENCES

[1] “Fundamentals of IoT and Wearable Technology Design | Wiley Online Books.” Accessed: Apr. 20, 2024. [Online]. Available: https://onlinelibrary-wiley-com.kuleuven.e-bronnen.be/doi/book/10.1002/9781119617570

[2] M. Chan, D. Estève, J.-Y. Fourniols, C. Escriba, and E. Campo, “Smart wearable systems: Current status and future challenges,” *Artif. Intell. Med.*, vol. 56, no. 3, pp. 137–156, Nov. 2012, doi: 10.1016/j.artmed.2012.09.003.

[3] K. Vix Kemanji, R. Mpwadina, and G. Meixner, “Virtual Reality Assembly of Physical Parts: The Impact of Interaction Interface Techniques on Usability and Performance,” in *Virtual, Augmented and Mixed Reality: Applications in Education, Aviation and Industry*, J. Y. C. Chen and G. Fragomeni, Eds., Cham: Springer International Publishing, 2022, pp. 350–368. doi: 10.1007/978-3-031-06015-1_24.

[4] J. Pan, D. Weng, and J. Mo, “Object Manipulation: Interaction for Virtual Reality on Multi-touch Screen,” in *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, Oct. 2019, pp. 280–284. doi: 10.1109/ISMAR-Adjunct.2019.00-29.

[5] “Springer Handbook of Augmented Reality | SpringerLink.” Accessed: Apr. 20, 2024. [Online]. Available: https://link-springer-com.kuleuven.e-bronnen.be/book/10.1007/978-3-030-67822-7

[6] M. Caeiro-Rodríguez, I. Otero-González, F. A. Mikic-Fonte, and M. Llamas-Nistal, “A Systematic Review of Commercial Smart Gloves: Current Status and Applications,” *Sensors*, vol. 21, no. 8, Art. no. 8, Jan. 2021, doi: 10.3390/s21082667.

[7] K. T. Lee, P. S. Chee, E. H. Lim, and Y. H. Kam, “Data Glove with Integrated Polyethylene-Carbon Composite-Based Strain Sensor for Virtual Reality Applications,” *Chem. Eng. Technol.*, vol. 46, no. 12, pp. 2480–2486, 2023, doi: 10.1002/ceat.202200569.

[8] M. Pan, Y. Tang, and H. Li, “State-of-the-Art in Data Gloves: A Review of Hardware, Algorithms, and Applications,” *IEEE Trans. Instrum. Meas.*, vol. 72, pp. 1–15, 2023, doi: 10.1109/TIM.2023.3243614.

[9] M. Pan, Y. Tang, and H. Li, “Review of the State-of-the-Art of Data Gloves,” in *2022 IEEE International Symposium on Smart Electronic Systems (iSES)*, Dec. 2022, pp. 402–405. doi: 10.1109/iSES54909.2022.00088.

[10] K. Grifantini, “AcceleGlove: Technology Review,” *Technol. Rev.*, vol. 112, no. 4, pp. 78–79, Aug. 2009.

[11] B. R. Glowacki and R. Freire, “An open source Etextile VR glove for real- time manipulation of molecular simulations”.

[12] “MTI | Free Full-Text | Hand-Controlled User Interfacing for Head-Mounted Augmented Reality Learning Environments.” Accessed: Apr. 20, 2024. [Online]. Available: https://www.mdpi.com/2414-4088/7/6/55

[13] Y. Han, “A low-cost visual motion data glove as an input device to interpret human hand gestures,” *IEEE Trans. Consum. Electron.*, vol. 56, no. 2, pp. 501–509, May 2010, doi: 10.1109/TCE.2010.5505962.

[14] J. Maitre, C. Rendu, K. Bouchard, B. Bouchard, and S. Gaboury, “Object recognition in performed basic daily activities with a handcrafted data glove prototype,” *Pattern Recognit. Lett.*, vol. 147, pp. 181–188, Jul. 2021, doi: 10.1016/j.patrec.2021.04.017.

[15] B. Ji *et al.*, “Flexible Strain Sensor-Based Data Glove for Gesture Interaction in the Metaverse: A Review,” *Int. J. Human–Computer Interact.*, vol. 0, no. 0, pp. 1–20, 2023, doi: 10.1080/10447318.2023.2212232.
[16] V. Schmücker, F. Grensing, R. Jakob, C. Gießer, R. Brück, and T. Eiler, “Conception And Implementation of an virtual Reality application for the evaluation of different types of commercially available haptic gloves,” *Curr. Dir. Biomed. Eng.*, vol. 9, no. 1, pp. 154–157, Sep. 2023, doi: 10.1515/cdbme-2023-1039.
[17] M. A. Ahmed, B. B. Zaidan, A. A. Zaidan, M. M. Salih, and M. M. bin Lakulu, “A Review on Systems-Based Sensory Gloves for Sign Language Recognition State of the Art between 2007 and 2017,” *Sensors*, vol. 18, no. 7, p. 2208, Jul. 2018, doi: 10.3390/s18072208.
[18] P. Achenbach, S. Laux, D. Purdack, P. N. Müller, and S. Göbel, “Give Me a Sign: Using Data Gloves for Static Hand-Shape Recognition,” *Sensors*, vol. 23, no. 24, p. 9847, Dec. 2023, doi: 10.3390/s23249847.
[19] E. A. Arkenbout, J. C. F. de Winter, and P. Breedveld, “Robust Hand Motion Tracking through Data Fusion of 5DT Data Glove and Nimble VR Kinect Camera Measurements,” *Sensors*, vol. 15, no. 12, pp. 31644–31671, Dec. 2015, doi: 10.3390/s151229868.
[20] C. Mizera, T. Delrieu, V. Weistroffer, C. Andriot, A. Decatoire, and J.-P. Gazeau, “Evaluation of Hand-Tracking Systems in Teleoperation and Virtual Dexterous Manipulation,” *IEEE Sens. J.*, vol. 20, no. 3, pp. 1642–1655, Feb. 2020, doi: 10.1109/JSEN.2019.2947612.
[21] C. M. J. Cox, B. Hicks, J. Gopsill, and C. Snider, “FROM HAPTIC INTERACTION TO DESIGN INSIGHT: AN EMPIRICAL COMPARISON OF COMMERCIAL HAND-TRACKING TECHNOLOGY,” *Proc. Des. Soc.*, vol. 3, pp. 1965–1974, Jul. 2023, doi: 10.1017/pds.2023.197.
[22] P. I. Morris and J. R. Rodriguez-Amat, “Collaborative Design in Kinetic Performance: Safeguarding the Uilleann Pipes through Inertial Motion Capture,” *Multimodal Technol. Interact.*, vol. 6, no. 11, Art. no. 11, Nov. 2022, doi: 10.3390/mti6110097.
[23] J. Li *et al.*, “Real-Time Hand Gesture Tracking for Human–Computer Interface Based on Multi-Sensor Data Fusion,” *IEEE Sens. J.*, vol. 21, no. 23, pp. 26642–26654, Dec. 2021, doi: 10.1109/JSEN.2021.3122236.
[24] “Motion capture company Rokoko launches Smartsuit Pro II,” *AnimationXpress*, Oct. 2021, Accessed: Apr. 30, 2024. [Online]. Available: https://www.proquest.com/docview/2587034894/citation/5A52E85DCB964DF5PQ/3
[25] M. van Wegen, “A novel virtual reality glove system with integrated vibro-tactile feedback for Parkinson’s disease: a usability study,” 2022, Accessed: Apr. 30, 2024. [Online]. Available: https://repository.tudelft.nl/islandora/object/uuid%3Aef85c5a4-0156-4ef4-a99e-ae5e08977248
[26] M. Caeiro-Rodríguez, I. Otero-González, F. A. Mikic-Fonte, and M. Llamas-Nistal, “A Systematic Review of Commercial Smart Gloves: Current Status and Applications,” *Sensors*, vol. 21, no. 8, Art. no. 8, Jan. 2021, doi: 10.3390/s21082667.
[27] admin_Hendri, “5DT Data Glove Ultra,” 5DT. Accessed: Apr. 30, 2024. [Online]. Available: https://5dt.com/5dt-data-glove-ultra/
[28] “Core API Introduction — SenseGlove Docs documentation.” Accessed: Apr. 30, 2024. [Online]. Available: https://senseglove.gitlab.io/SenseGloveDocs/native/core-api-intro.html
[29] “MANUS.” Accessed: Apr. 30, 2024. [Online]. Available: https://www.manus-meta.com/knowledge-products/plugins
[30] “Integrate Rokoko mocap tools in real-time to Blender, Unreal, Unity and more.” Accessed: Apr. 30, 2024. [Online]. Available: https://www.rokoko.com/integrations

[31] “Sdk – Sensoryx.” Accessed: Apr. 30, 2024. [Online]. Available: https://sensoryx.tech/sdk/
[32] Kang Li, I-Ming Chen, Song Huat Yeo, and Chee Kian Lim, “Development of finger-motion capturing device based on optical linear encoder: Journal of Rehabilitation Research & Development,” *J. Rehabil. Res. Dev.*, vol. 48, no. 1, pp. 69–82, Jan. 2011, doi: 10.1682/JRRD.2010.02.0013.
[33] M. Hosseini, Y. Pane, A. Sengül, J. De Schutter, and H. Bruyninckx, “A Novel Haptic Glove (ExoTen-Glove) Based on Twisted String Actuation (TSA) System for Virtual Reality,” in *Haptics: Science, Technology, and Applications*, D. Prattichizzo, H. Shinoda, H. Z. Tan, E. Ruffaldi, and A. Frisoli, Eds., Cham: Springer International Publishing, 2018, pp. 612–622. doi: 10.1007/978-3-319-93399-3_52.
[34] B.-S. Lin, I.-J. Lee, P.-Y. Chiang, S.-Y. Huang, and C.-W. Peng, “A Modular Data Glove System for Finger and Hand Motion Capture Based on Inertial Sensors,” *J. Med. Biol. Eng.*, vol. 39, no. 4, pp. 532–540, Aug. 2019, doi: 10.1007/s40846-018-0434-6.
[35] Y. Li *et al.*, “Low-Cost Data Glove Based on Deep-Learning-Enhanced Flexible Multiwalled Carbon Nanotube Sensors for Real-Time Gesture Recognition,” *Adv. Intell. Syst.*, vol. 4, no. 11, p. 2200128, 2022, doi: 10.1002/aisy.202200128.
[36] C. Wu *et al.*, “Development of a Low-Cost Wearable Data Glove for Capturing Finger Joint Angles,” *Micromachines*, vol. 12, no. 7, Art. no. 7, Jul. 2021, doi: 10.3390/mi12070771.
[37] W. Lin, C. Li, and Y. Zhang, “Interactive Application of Data Glove Based on Emotion Recognition and Judgment System,” *Sensors*, vol. 22, no. 17, p. 6327, Aug. 2022, doi: 10.3390/s22176327.
[38] L. Dipietro, A. M. Sabatini, and P. Dario, “A Survey of Glove-Based Systems and Their Applications,” *IEEE Trans. Syst. Man Cybern. Part C Appl. Rev.*, vol. 38, no. 4, pp. 461–482, Jul. 2008, doi: 10.1109/TSMCC.2008.923862.
[39] “fusionTrack 500 – Atracsys Measurement Solutions.” Accessed: May 02, 2024. [Online]. Available: https://atracsys.com/fusiontrack-500/
[40] *Virtual and Augmented Reality (VR/AR)*. Accessed: May 01, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-3-030-79062-2
[41] D.-C. Hoang, A. J. Lilienthal, and T. Stoyanov, “Object-RPE: Dense 3D reconstruction and pose estimation with convolutional neural networks,” *Robot. Auton. Syst.*, vol. 133, p. 103632, Nov. 2020, doi: 10.1016/j.robot.2020.103632.
[42] J. Kuti, T. Piricz, and P. Galambos, “Method for Direction and Orientation Tracking Using IMU Sensor,” *IFAC-Pap.*, vol. 56, no. 2, pp. 10774–10780, Jan. 2023, doi: 10.1016/j.ifacol.2023.10.744.
[43] K. Park, S. Kim, Y. Yoon, T.-K. Kim, and G. Lee, “DeepFisheye: Near-Surface Multi-Finger Tracking Technology Using Fisheye Camera,” in *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, in UIST ’20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 1132–1146. doi: 10.1145/3379337.3415818.
[44] H. Zhou, T. Lu, Y. Liu, S. Zhang, and M. Gowda, “Learning on the Rings: Self-Supervised 3D Finger Motion Tracking Using Wearable Sensors,” *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 6, no. 2, p. 90:1-90:31, Jul. 2022, doi: 10.1145/3534587.
[45] F. S. Parizi, E. Whitmire, and S. Patel, “AuraRing: Precise Electromagnetic Finger Tracking,” *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, vol. 3, no. 4, p. 150:1-150:28, Sep. 2020, doi: 10.1145/3369831.
[46] N. Tongrod, S. Lokavee, N. Watthanawisuth, A. Tuantranont, and T. Kerdcharoen, “Design and development of data glove based on printed polymeric sensors and

Zigbee networks for Human–Computer Interface," *Disabil. Rehabil. Assist. Technol.*, vol. 8, no. 2, pp. 115–120, Mar. 2013, doi: 10.3109/17483107.2012.737540.
[47] "Internet of Things (IoT) communication protocols: Review | IEEE Conference Publication | IEEE Xplore." Accessed: Apr. 20, 2024. [Online]. Available: https://ieeexplore-ieee-org.kuleuven.e-bronnen.be/abstract/document/8079928
[48] G. Gardašević, K. Katzis, D. Bajić, and L. Berbakov, "Emerging Wireless Sensor Networks and Internet of Things Technologies—Foundations of Smart Healthcare," *Sensors*, vol. 20, no. 13, Art. no. 13, Jan. 2020, doi: 10.3390/s20133619.
[49] V. S. Chakravarthi, "M2M Communication and Technologies," in *Internet of Things and M2M Communication Technologies: Architecture and Practical Design Approach to IoT in Industry 4.0*, V. S. Chakravarthi, Ed., Cham: Springer International Publishing, 2021, pp. 151–166. doi: 10.1007/978-3-030-79272-5_10.
[50] L. E. Frenzel, "Chapter Two - Serial I/O Primer: A Short Course in Data Communications and Networking," in *Handbook of Serial Communications Interfaces*, L. E. Frenzel, Ed., Oxford: Newnes, 2016, pp. 5–27. doi: 10.1016/B978-0-12-800629-0.00002-4.
[51] C. Antón-Haro and M. Dohler, "1 - Introduction to machine-to-machine (M2M) communications," in *Machine-to-machine (M2M) Communications*, C. Antón-Haro and M. Dohler, Eds., Oxford: Woodhead Publishing, 2015, pp. 1–23. doi: 10.1016/B978-1-78242-102-3.00001-0.
[52] R. Jordan and C. T. Abdallah, "Wireless communications and networking: an overview," *IEEE Antennas Propag. Mag.*, vol. 44, no. 1, pp. 185–193, Feb. 2002, doi: 10.1109/74.997963.
[53] *Massive Machine Type Communications*. Accessed: Apr. 27, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-3-030-13574-4
[54] E. Gupta, S. S. S. Valisetti, and P. Madhavan, "The Smart Glasses Using Iot And Unity," *Turk. J. Comput. Math. Educ.*, vol. 12, no. 11, pp. 4823–4830, 2021.
[55] J. Pernas-Álvarez and D. Crespo-Pereira, "Open-source 3D discrete event simulator based on the game engine unity," *J. Simul.*, vol. 0, no. 0, pp. 1–17, 2024, doi: 10.1080/17477778.2024.2314166.
[56] B. Wu, S. Zhang, W. Tian, and H. Wang, "Application and Development Prospect of Monitoring Screen based on Three.js Unit Equipment Control System," in *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)*, Dec. 2022, pp. 347–351. doi: 10.1109/QRS-C57518.2022.00058.
[57] I. Malfait and T. Vandenbroecke, "Flutter and other multi-platform software development kits practically applied and theoretically compared".
[58] "AR and VR Using the WebXR API: Learn to Create Immersive Content with WebGL, Three.js, and A-Frame | SpringerLink." Accessed: Apr. 20, 2024. [Online]. Available: https://link-springer-com.kuleuven.e-bronnen.be/book/10.1007/978-1-4842-6318-1
[59] *Practical Flutter*. Accessed: Apr. 20, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-4972-7
[60] K. De Burghgraeve, "Pose estimation for AGVs: a comparison between ArUco markers and HTC Vive using ROS2," KU Leuven. Faculteit Industriële Ingenieurswetenschappen, Leuven, 2022.
[61] M. Summerfield, *Rapid GUI programming with Python and Qt: the definitive guide to PyQt programming.* in Prentice Hall open source software development series. Upper Saddle River: Prentice Hall, 2008.
[62] K. Vix Kemanji, R. Mpwadina, and G. Meixner, "Virtual Reality Assembly of Physical Parts: The Impact of Interaction Interface Techniques on Usability and Performance," in *Virtual, Augmented and Mixed Reality: Applications in Education, Aviation and*

Industry, J. Y. C. Chen and G. Fragomeni, Eds., Cham: Springer International Publishing, 2022, pp. 350–368. doi: 10.1007/978-3-031-06015-1_24.
[63] A. Venkataraman and K. K. Jagadeesha, “Evaluation of Inter-Process Communication Mechanisms”.
[64] X. Meng, S. Zhao, Y. Huang, Z. Zhang, J. Eagan, and R. Subramanian, “WADE: simplified GUI add-on development for third-party software,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, in CHI ’14. New York, NY, USA: Association for Computing Machinery, Apr. 2014, pp. 2221–2230. doi: 10.1145/2556288.2557349.
[65] *Pro REST API Development with Node.js*. Accessed: Apr. 20, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-0917-2
[66] *Embedded and Real-Time Operating Systems*. Accessed: Apr. 28, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-3-031-28701-5
[67] *Accelerating Development Velocity Using Docker*. Accessed: Apr. 20, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-3936-0
[68] “Using the ESP32 Microcontroller for Data Processing | IEEE Conference Publication | IEEE Xplore.” Accessed: Apr. 20, 2024. [Online]. Available: https://ieeexplore-ieee-org.kuleuven.e-bronnen.be/document/8765944
[69] C. Melançon, G. Simard, M. Saad, K. Kaur, and J. Gascon-Samson, “BlazeFlow: a Multi-Layer Communication Middleware for Real-Time Distributed IoT Applications,” in *Proceedings of the 1st International Workshop on Middleware for the Computing Continuum*, in Mid4CC ’23. New York, NY, USA: Association for Computing Machinery, Dec. 2023, pp. 30–35. doi: 10.1145/3631309.3632837.
[70] A. H. Ngu, M. Gutierrez, V. Metsis, S. Nepal, and Q. Z. Sheng, “IoT Middleware: A Survey on Issues and Enabling Technologies,” *IEEE Internet Things J.*, vol. 4, no. 1, pp. 1–20, Feb. 2017, doi: 10.1109/JIOT.2016.2615180.
[71] R. Estrada and I. Ruiz, “The Broker: Apache Kafka,” in *Big Data SMACK: A Guide to Apache Spark, Mesos, Akka, Cassandra, and Kafka*, R. Estrada and I. Ruiz, Eds., Berkeley, CA: Apress, 2016, pp. 165–203. doi: 10.1007/978-1-4842-2175-4_8.
[72] T. Chakraborti, S. Srivastava, A. Pinto, and S. Kambhampati, “An ROS-based Shared Communication Middleware for Plug & Play Modular Intelligent Design of Smart Systems,” *arXiv.org*, Jun. 2017, Accessed: Apr. 20, 2024. [Online]. Available: https://www.proquest.com/docview/2075644272?pq-origsite=primo&sourcetype=Working%20Papers
[73] *Robot Operating System (ROS)*. Accessed: Apr. 20, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-3-031-09062-2
[74] alldatasheet.com, “ATMEGA328P Datasheet(PDF).” Accessed: May 01, 2024. [Online]. Available: http://www.alldatasheet.com/datasheet-pdf/pdf/241077/ATMEL/ATMEGA328P.html
[75] R. P. Ltd, “RP2040 specifications,” Raspberry Pi. Accessed: May 01, 2024. [Online]. Available: https://www.raspberrypi.com/products/rp2040/
[76] “ESP32-WROOM-32E/32UE | Espressif Systems.” Accessed: May 01, 2024. [Online]. Available: https://www.espressif.com/en/producttype/esp32-wroom-32e32ue
[77] “LSM6DSO32X - iNEMO inertial module: always-on 3D accelerometer and 3D gyroscope - STMicroelectronics.” Accessed: May 01, 2024. [Online]. Available: https://www.st.com/en/mems-and-sensors/lsm6dso32x.html
[78] *The Earth’s Magnetism*. Accessed: May 01, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-3-540-27980-8
[79] alldatasheet.com, “MMC5603NJ Datasheet(PDF).” Accessed: May 01, 2024. [Online]. Available: http://www.alldatasheet.com/datasheet-pdf/pdf/1361261/MEMSIC/MMC5603NJ.html

[80] M. Kok and T. B. Schön, “Magnetometer calibration using inertial sensors,” *IEEE Sens. J.*, vol. 16, no. 14, pp. 5679–5689, Jul. 2016, doi: 10.1109/JSEN.2016.2569160.
[81] M. Nazarahari and H. Rouhani, “Sensor fusion algorithms for orientation tracking via magnetic and inertial measurement units: An experimental comparison survey,” *Inf. Fusion*, vol. 76, pp. 8–23, Dec. 2021, doi: 10.1016/j.inffus.2021.04.009.
[82] “Estimation of IMU and MARG orientation using a gradient descent algorithm | IEEE Conference Publication | IEEE Xplore.” Accessed: May 01, 2024. [Online]. Available: https://ieeexplore-ieee-org.kuleuven.e-bronnen.be/document/5975346
[83] “Internet of Things (IoT) communication protocols: Review | IEEE Conference Publication | IEEE Xplore.” Accessed: Apr. 20, 2024. [Online]. Available: https://ieeexplore-ieee-org.kuleuven.e-bronnen.be/abstract/document/8079928
[84] G. Gardašević, K. Katzis, D. Bajić, and L. Berbakov, “Emerging Wireless Sensor Networks and Internet of Things Technologies—Foundations of Smart Healthcare,” *Sensors*, vol. 20, no. 13, Art. no. 13, Jan. 2020, doi: 10.3390/s20133619.
[85] C. Antón-Haro and M. Dohler, “1 - Introduction to machine-to-machine (M2M) communications,” in *Machine-to-machine (M2M) Communications*, C. Antón-Haro and M. Dohler, Eds., Oxford: Woodhead Publishing, 2015, pp. 1–23. doi: 10.1016/B978-1-78242-102-3.00001-0.
[86] “Fundamentals of IoT and Wearable Technology Design | Wiley Online Books.” Accessed: Apr. 20, 2024. [Online]. Available: https://onlinelibrary-wiley-com.kuleuven.e-bronnen.be/doi/book/10.1002/9781119617570
[87] V. S. Chakravarthi, “M2M Communication and Technologies,” in *Internet of Things and M2M Communication Technologies: Architecture and Practical Design Approach to IoT in Industry 4.0*, V. S. Chakravarthi, Ed., Cham: Springer International Publishing, 2021, pp. 151–166. doi: 10.1007/978-3-030-79272-5_10.
[88] L. E. Frenzel, “Chapter Three - Selecting an Appropriate Interface,” in *Handbook of Serial Communications Interfaces*, L. E. Frenzel, Ed., Oxford: Newnes, 2016, pp. 29–31. doi: 10.1016/B978-0-12-800629-0.00003-6.
[89] N. Cameron, “ESP32 Microcontroller,” in *ESP32 Formats and Communication: Application of Communication Protocols with ESP32 Microcontroller*, N. Cameron, Ed., Berkeley, CA: Apress, 2023, pp. 1–54. doi: 10.1007/978-1-4842-9376-8_1.
[90] “Springer Handbook of Augmented Reality | SpringerLink.” Accessed: Apr. 20, 2024. [Online]. Available: https://link-springer-com.kuleuven.e-bronnen.be/book/10.1007/978-3-030-67822-7
[91] “MTI | Free Full-Text | Hand-Controlled User Interfacing for Head-Mounted Augmented Reality Learning Environments.” Accessed: Apr. 20, 2024. [Online]. Available: https://www.mdpi.com/2414-4088/7/6/55
[92] *UX for XR*. Accessed: Apr. 20, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-7020-2
[93] “The Haply Development Platform | Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems.” Accessed: Apr. 20, 2024. [Online]. Available: https://dl-acm-org.kuleuven.e-bronnen.be/doi/abs/10.1145/3170427.3186512
[94] J. He, X. Chen, S. Zhang, X. Zhang, W. Kong, and T. Zhang, “Software for Wearable Devices: Challenges and Opportunities,” *arXiv.org*, Apr. 2015, Accessed: Apr. 20, 2024. [Online]. Available: https://www.proquest.com/docview/2081375779?parentSessionId=1GYDgGbzV9LDYwm33C1w7RD74osNiNULVu0M48uktgM%3D&parentSessionId=JJDQxfwg5iUnfPyWjcoOdu9s3Yf8J1mrVzSUs9hPIhk%3D&parentSessionId=UEUQfU%2BBWrgZ3IKffbCVJuhD69U7eoqAlGJObd8v2x0%3D&pq-origsite=primo&sourcetype=Working%20Papers

[95] U. Technologies, “Start Your Creative Projects and Download the Unity Hub | Unity.” Accessed: May 03, 2024. [Online]. Available: https://unity.com/download
[96] E. Ćiković, K. Mäusl, and K. Lenac, “Virtual Reality Applications with Oculus Rift and 3D Sensors,” in *Augmented Reality, Virtual Reality, and Computer Graphics*, L. T. De Paolis and A. Mongelli, Eds., Cham: Springer International Publishing, 2016, pp. 181–185. doi: 10.1007/978-3-319-40621-3_13.
[97] J. Pernas-Álvarez and D. Crespo-Pereira, “Open-source 3D discrete event simulator based on the game engine unity,” *J. Simul.*, vol. 0, no. 0, pp. 1–17, 2024, doi: 10.1080/17477778.2024.2314166.
[98] A. B. Lima and E. Correa, “Graphical Visualization on Computational Simulation Using Shared Memory,” *J. Phys. Conf. Ser.*, vol. 487, no. 1, p. 012014, Mar. 2014, doi: 10.1088/1742-6596/487/1/012014.
[99] “AR and VR Using the WebXR API: Learn to Create Immersive Content with WebGL, Three.js, and A-Frame | SpringerLink.” Accessed: Apr. 20, 2024. [Online]. Available: https://link-springer-com.kuleuven.e-bronnen.be/book/10.1007/978-1-4842-6318-1
[100] “ICT Infrastructure and Computing: Proceedings of ICT4SD 2023, Volume 3 | SpringerLink.” Accessed: Apr. 20, 2024. [Online]. Available: https://link-springer-com.kuleuven.e-bronnen.be/book/10.1007/978-981-99-4932-8
[101] *Practical Flutter*. Accessed: Apr. 20, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-1-4842-4972-7
[102] M. Summerfield, *Rapid GUI programming with Python and Qt: the definitive guide to PyQt programming.* in Prentice Hall open source software development series. Upper Saddle River: Prentice Hall, 2008.
[103] H. Malallah *et al.*, “A Comprehensive Study of Kernel (Issues and Concepts) in Different Operating Systems,” *Asian J. Res. Comput. Sci.*, pp. 16–31, May 2021, doi: 10.9734/ajrcos/2021/v8i330201.
[104] “Embedded Operating Systems: A Practical Approach | SpringerLink.” Accessed: Apr. 20, 2024. [Online]. Available: https://link-springer-com.kuleuven.e-bronnen.be/book/10.1007/978-3-319-72977-0
[105] *Robot Operating System (ROS)*. Accessed: Apr. 20, 2024. [Online]. Available: https://link.springer.com/book/10.1007/978-3-031-09062-2
[106] K. Chinnappan, I. Malavolta, G. A. Lewis, M. Albonico, and P. Lago, “Architectural Tactics for Energy-Aware Robotics Software: A Preliminary Study,” in *Software Architecture*, S. Biffl, E. Navarro, W. Löwe, M. Sirjani, R. Mirandola, and D. Weyns, Eds., Cham: Springer International Publishing, 2021, pp. 164–171. doi: 10.1007/978-3-030-86044-8_11.
[107] “Vulcanexus, the All-in-One ROS 2 tool set,” Vulcanexus. Accessed: May 04, 2024. [Online]. Available: https://vulcanexus.org/
[108] J. Zhang, F. Keramat, X. Yu, D. M. Hern, J. P. Queralta, and T. Westerlund, “Distributed Robotic Systems in the Edge-Cloud Continuum with ROS 2: a Review on Novel Architectures and Technology Readiness.” arXiv, Nov. 02, 2022. Accessed: Apr. 22, 2024. [Online]. Available: http://arxiv.org/abs/2211.00985
[109] E. Coronado and G. Venture, “Towards IoT-Aided Human–Robot Interaction Using NEP and ROS: A Platform-Independent, Accessible and Distributed Approach,” *Sensors*, vol. 20, no. 5, p. 1500, Mar. 2020, doi: 10.3390/s20051500.
[110] “Foxglove - Visualizing and debugging your robotics data,” Foxglove. Accessed: May 04, 2024. [Online]. Available: https://foxglove.dev/
[111] “Analysis of protocols used for visualization in automotive industry | IEEE Conference Publication | IEEE Xplore.” Accessed: Apr. 20, 2024. [Online]. Available: https://ieeexplore-ieee-org.kuleuven.e-bronnen.be/document/10174226

[112] G. Toffetti, L. Militano, R. Tharaka, and M. Straub, “ROS-based Robotic Applications Orchestration in the Compute Continuum: Challenges and Approaches,” in *Proceedings of the IEEE/ACM 16th International Conference on Utility and Cloud Computing*, in UCC ’23. New York, NY, USA: Association for Computing Machinery, Apr. 2024, pp. 1–6. doi: 10.1145/3603166.3632555.

[113] D. Shor *et al.*, “Designing Haptics: Improving a Virtual Reality Glove with Respect to Realism, Performance, and Comfort,” *Int. J. Autom. Technol.*, vol. 13, no. 4, pp. 453–463, Jul. 2019, doi: 10.20965/ijat.2019.p0453.

8 APPENDIX

The final GitHub repository comprised of all software and hardware files can be found at the following GitHub repository:

https://github.com/honza1r/DataGlove-Thesis

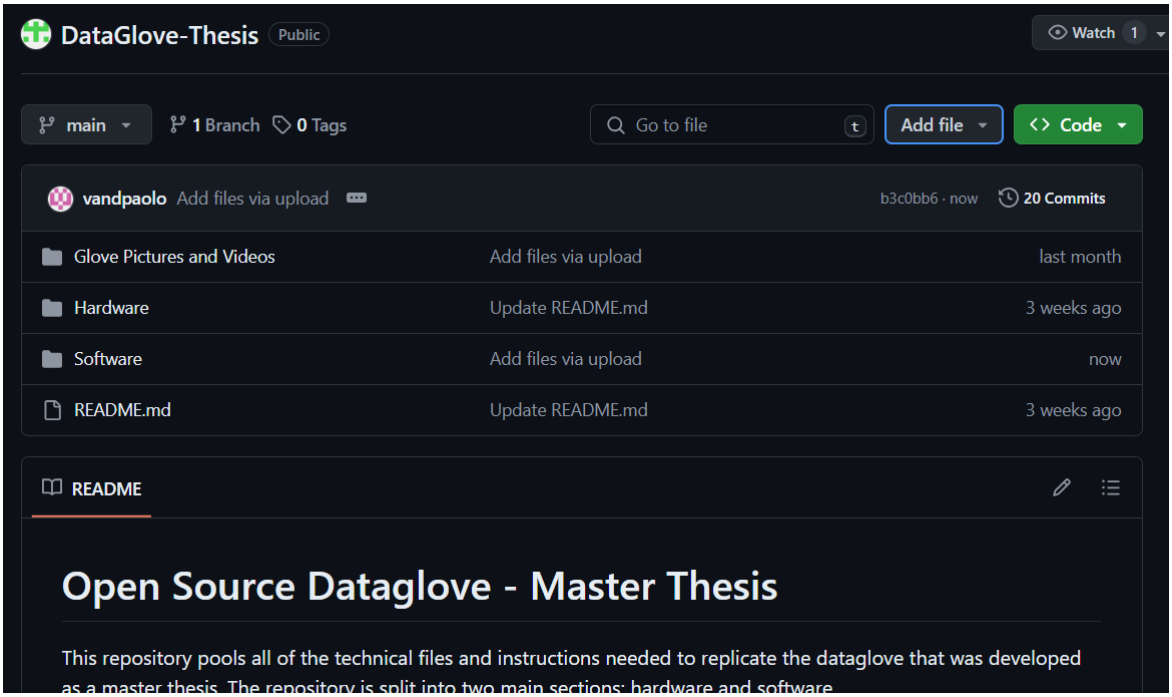

Figure 29: GitHub Repository Files

The following files are contained within the repository:

Glove Pictures and Videos: Video and photo material taken throughout the thesis of the development, the working prototypes and simulation within GUI.

Hardware: All necessary hardware files, PCB schematics, calibration scripts, and necessary Arduino firmwares including EKF and Madgwick filters.

Software: Docker based environment. Easily compiled following guidelines after necessary changes to IP. Tested on Docker running on WSL runtime on Windows 11

FACULTY OF ENGINEERING TECHNOLOGY
GROUP T LEUVEN CAMPUS
Andreas Vesaliusstraat 13
3000 LEUVEN, België
tel. + 32 16 30 10 30
fet.groupt@kuleuven.be
www.fet.kuleuven.be