Delft University of Technology
Master of Science Thesis in Embedded Systems

Ultra Low Latency Object Tracking for Tactile Internet

Berkin Zeybekoglu

Networked Systems

Ultra Low Latency Object Tracking for Tactile Internet

Master of Science Thesis in Embedded Systems

Networked Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

Berkin Zeybekoglu

October 21st 2024

Author
Berkin Zeybekoglu
Title
Ultra Low Latency Object Tracking for Tactile Internet
MSc Presentation Date
November 4th 2024

Graduation Committee

Dr. Ranga Rao Venkatesha Prasad	Delft University of Technology
Dr. Avishek Anand	Delft University of Technology
Kees Kroep	Delft University of Technology

Abstract

Bilateral teleoperation with force feedback aims to transmit human expertise over long distances by transferring the sensation of physical contact. One of the primary challenges in achieving this goal is the ultra low latency requirement. Tactile internet and model-mediated teleoperation are promising research areas addressing the latency constraint. In model-mediated teleoperation, it is crucial that the remote environment parameters are tracked with minimal latency to update the local model. This work investigates the potential of designing a cost-effective object tracking solution that performs comparably to state-of-the-art systems while maintaining low tracking latency. A method is proposed to streamline the design process by leveraging assumptions about the tracking conditions. An object-tracking algorithm that can effectively fuse high-frequency noisy inertial data (>1 kHz) with delayed, low-frequency (30 Hz) but accurate camera data has been designed. A practical system was constructed using cost-efficient components, and a dataset was collected for testing and characterization. The system demonstrated the ability to produce object pose estimates at 1 ms intervals. In experiments along a single axis, the system achieved a mean positional estimation error of 0.1 cm and a mean orientation estimation error of 0.5°. The algorithm also successfully corrected for camera latencies of up to 350 ms while maintaining accurate estimates. These results demonstrate the possibility of achieving a low latency, accurate object tracking system while keeping component and computation costs reasonable.

Acknowledgements

Both the road leading to this project and the project itself have been a long journey with its ups and downs, and I could not have made it without the support of the many people around me along the way.

I would like to thank my friends Karan Pathak, Ismail Bourhaeil, and Tomás Hernández Quintanilla, with whom I have worked closely many times throughout this master's journey, from whom I have learned invaluable lessons and whose camaraderie I have greatly enjoyed. I would like to thank David Zwart for sharing his knowledge and resources on 3D printing. His help has significantly sped up the process and improved the quality of the printed objects used in this project. Many thanks to Stijn Coppens. His expertise with the robot arm has helped set up and execute significant parts of this project, and his company during office days was much appreciated.

My deep gratitude goes to my daily co-supervisor, Kees Kroep. His creativity and support have given me the weekly boosts I needed to complete this project. I am grateful for his assistance and patience.

I would like to thank Dr. Ranga Rao Venkatesha Prasad for his guidance and for giving me the chance to work with his group on this project.

I would like to thank my friends for their encouragement and companionship. Finally, my profound appreciation to my dear family. They have supported me unconditionally through thick and thin; I could not have done this without them.

Berkin Zeybekoglu

Delft, The Netherlands
October 21st 2024

Contents

Chapter 1

Introduction

In today's world, connecting with someone across the globe is as easy as a single click. Even with a standard smartphone, we can see and hear the person we are talking to, enjoying a natural, real-time conversation with minimal effort. Since their inception, cell phones and video calling technologies have overcome many communication barriers and have been essential in simplifying complex processes in the industry and our daily lives.

However, as these technologies approach a state of refinement, it raises the question of what the next evolutionary step in human interaction will be. We argue that the answer lies in remote force-feedback interactions, specifically bilateral teleoperation with force feedback.

Bilateral teleoperation with force feedback enables the possibility to physically manipulate a remote environment while simultaneously preserving the feeling of the applied forces locally. A compelling example of this kind of interaction might be found in disaster response and hazardous environment exploration. In an earthquake disaster scenario where rescue teams are tasked with finding survivors in the rubble, teleoperated robots can be used for the search mission instead of sending in humans. They can navigate through dangerous and unstable terrain. The operators will be able to feel the resistance of the materials and their textures in real time. This enables them to decide the appropriate levels of force to remove the debris and will effectively increase the safety, efficiency, and success of the extraction of survivors. This might be an example of a somewhat extreme case, but bilateral teleoperation will leave no industry unaffected, from gaming to agriculture and from the medical industry to education.

Individual industries aside, one of the most prominent changes this technology will bring is the effortless importation and exportation of human skills worldwide and beyond. Experts from any field that requires human physical interaction can lend their expertise without moving significantly large distances. This has many palpable advantages but also accommodates many benefits that are not evident initially. Professionals who do not have to relocate to offer their expertise conserve ample resources when considered in scale. This approach saves vast amounts of time, money, and energy, contributing to individual and travel-related carbon emissions goals. Furthermore, remote access to expertise can create unprecedented opportunities for smaller and isolated communities by providing previously unreachable services and skills. This can even boost local economic development and provide a more equal distribution of resources. The distribution of economic activities more evenly across different regions by making living and working conditions outside of major urban centers can help address the problem of overpopulation from which many large cities worldwide suffer. This technology's environmental, economic, and social advantages promote a more sustainable approach to managing the global workforce, enabling broader access to opportunities for everyone.

Obviously, this future is still far ahead of us, but research on bilateral teleoperation with force feedback has existed for many years. In the classic sense, a bilateral teleoperation system consists of two distinct domains, the operator domain and the remote domain, and

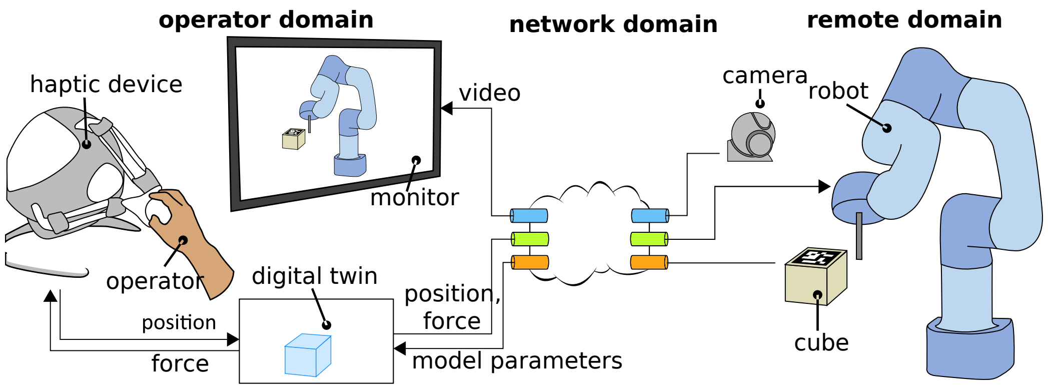


Figure 1.1: **Illustration of a model-mediated bilateral teleoperation system.**

a network that connects these two domains. An example of a bilateral teleoperation setup can be seen in Figure 1.1. The operator domain consists of a haptic device with which an operator can interact. The haptic device is connected to a robotic actuator in the remote domain. The robotic actuator imitates the movements that the operator dictates on the haptic device and, thus, interacts with its environment. The interaction of the robotic actuator is captured via sensors and sent back to the operator domain over the network, where the force feedback can be applied to the haptic device such that the operator can feel the consequences of its remote manipulations. In addition to the force-feedback in some bilateral teleoperation settings, a live audio-visual feed of the remote environment can be incorporated. This feature, in combination with precise force feedback, can be crucial in scenarios where the operator has to maintain accurate hand-eye coordination.

A significant challenge in realizing bilateral teleoperation systems is managing the delay that is introduced by the network and other system components. When interacting with a remote environment, it is crucial that the applied force is relayed back to the operator in time. The presence of significant lag, resulting in delayed feedback, can lead to the operator applying too much force, or the force can be applied for too long. Especially in critical applications, this can have detrimental results. Research has found that the time of a round trip latency for feedback must be in the order of 1 ms [1]. This is also called the ultra low latency requirement. One approach that deals with this requirement is Tactile Internet (TI). TI aims to minimize network latencies as much as possible, which can enable reliable low-latency bilateral teleoperation. The foremost obstacle in applying TI over long distances is the fundamental limitation imposed by the speed of light. Light can traverse a maximum distance of approximately 300 kilometers (or 150 km each way) before exceeding the 1 ms latency threshold. Another approach that tries to accomplish reliable and stable bilateral teleoperation is model-mediated teleoperation (MMT). MMT considers that there will always be a latency component present in such systems and tries to deal with this problem by introducing a model of the remote domain that runs in the operator domain. This local model is a digital twin of the remote domain. When the operator interacts with the haptic device, the force feedback can be calculated almost instantly through the local model, and it can be applied before the actual feedback from the remote environment even arrives. This local model enables a smooth teleoperation experience for the operator while handling divergence and latency issues under the hood. MMT research has predominantly focused on static remote environments. This indicates that objects in the remote environment cannot be manipulated such that they undergo large displacement, rotation, or deformation. Under these static circumstances, parameters to form the local model need to be sent over the network just once, and therefore, only minor corrections might be needed during operation. The real challenge arises when dynamic environments are considered. Dynamic environments inherently need a multitude more parameters to form an accurate model. If the environment is not known beforehand, environment detection is needed. Furthermore, since the parameters of the objects will change significantly during operation in such environments, the new parameters must be calculated and sent back to the operator domain to update the local model. This considerably increases the system's complexity as it requires updating the local model frequently and more drastically. Hence, the need for a reliable, accurate, robust object-tracking system arises. State-of-the-art object tracking methods are reliable but often are costly solutions, require complex setups, and suffer from computational limits. To enable and kickstart accessible research on bilateral teleoperation, an object-tracking system that solves these problems is needed, which brings us to the research question of this thesis:

How can an accurate, low-latency object pose tracking system be set up as accessible as possible with minimal component and time costs?

This thesis proposes a method for reliably tracking an object with low latency. It describes an accessible system implementation that can be used to research bilateral teleoperation systems with relative ease and minimal cost. Our approach utilizes both inertial sensor data and camera data, leveraging the advantages of each method. We employ a custom pose-tracking algorithm to fuse data from both sensors. The practical system setup is implemented using off-the-shelf components and validated with a robot arm setup. Our focus is on creating a system that is lightweight, easy to set up, user-friendly, and cost-effective.

The contributions of this thesis are as follows:

- Proposed a method to tackle the object tracking component in bilateral teleoperation research in a practical and accessible way.
- Designed an object tracking algorithm that accurately estimates object pose with low latency by fusing inertial measurements with camera data.
- Designed and built a cost-effective object tracking system that integrates the proposed algorithm into a practical setup.
- Built a dataset with reliable ground truth data using a robot arm with accurate pathing.
- Demonstrated the effectiveness of the proposed method by analyzing and characterizing the proposed system with the produced dataset.

The rest of the report is organized as follows: chapter 2 reviews the relevant literature accumulated over the years related to this research, chapter 3 explains the theoretical concepts that are of importance to understand the designed system and the work done, chapter 4 describes the methods employed in this thesis, chapter 5 states the implementation details of the designed system, chapter 6 discusses and analyses the obtained results, chapter 7 explains steps that could be taken in further research, chapter 8 evaluates the project process and learning outcomes, and finally, chapter 9 finishes this report with concluding remarks.

Chapter 2

Related Works

2.1 Model Mediated Teleoperation

Model-mediated teleoperation (MMT) is a technique used in bilateral teleoperation systems to enhance stability and performance, particularly in environments with significant communication delays. This technique uses a local model of the remote environment to measure and provide force feedback to the user, eliminating the need to wait for relayed signals from the remote side. The concept of using local models to improve stability in bilateral teleoperation applications has been around since the late 1980s [2, 3, 4]. Over the years, research in this area has advanced significantly, and [5] has introduced the contemporary concept of MMT. Until this point, the remote environments considered in these applications were static. This was due to the increased complexity of tracking parameters required to update local models in dynamic environments. Consequently, no sophisticated methods were required to track the model parameters in the remote environment. However, in the early 2010s, research began addressing the challenges of dynamic remote environments [6, 7]. For instance, in [6], the authors use a combination of position, force, and vision sensors to obtain a system with 2 degrees of freedom (DoF). Similarly, in [7], a stereo camera setup is combined with point clouds to track and update the model parameters of the tracked environment. Additionally, [8] can be considered the first comprehensive study including actual movable objects in the environment. The authors consider a 3 DoF system, but no specific tracking method is proposed. Today, research on MMT continues to evolve [9, 10].

2.2 Object Pose Tracking

The position and orientation of an object in 3D space together form the object's pose. Estimating and tracking the pose of one or multiple objects is essential in various fields, including augmented reality (AR) [11], virtual reality (VR) [12], autonomous driving [13], and robotics [14]. Each of these applications has unique requirements and challenges. For instance, autonomous driving systems require a high degree of accuracy and robustness to ensure the safety and reliability of the vehicle in various environmental conditions [15]. On the other hand, AR and VR systems often demand real-time capabilities with low latency to provide a seamless and immersive user experience. In these scenarios, even slight delays can break the illusion and cause discomfort to the user [16]. Similarly, model mediation with dynamic environments in bilateral teleoperation has its own unique set of requirements that must be fulfilled. Precise and real-time tracking of object pose is essential to ensure accurate and responsive control, which is crucial for tasks requiring high levels of dexterity and coordination. Pose tracking algorithms for these systems demand low latency, high-frequency data updates, robust handling of variable environmental conditions, and often working with delayed measurements.

2.2.1 Visual Pose Tracking

One field that has particularly been interested in the object pose estimation and tracking problem for a long time is computer vision. Although this problem has been studied for a long time in this field, developments in computing technology, the advancement of machine learning methods, and the availability of much higher quality cameras in recent years have resulted in rapid advancement in pose tracking algorithms [17]. The authors of [18] divide visual object pose tracking methods roughly into three categories: traditional methods, end-to-end deep-learning-based methods, and a combination of the two.

Traditional methods are robust, effective in controlled environments, and generally computationally efficient. Nevertheless, they tend to suffer from problems like background clutter, lighting conditions, and occlusion. Furthermore, they are less flexible in dealing with unseen objects as the tracked objects need to be registered beforehand, either as features or as a template of the object. These algorithms generally operate in two stages where the features from an image are matched with object features using feature detection and description algorithms like RANSAC [19] and SIFT [20]. Then, the pose of the object is estimated using an algorithm like Perspective-n-Point (PnP) [19, 21], which uses non-linear optimization to reproject the matched 2D features into 3D space. [22, 23, 24] are some recent works in this category.

The end-to-end deep-learning methods provide high accuracy and robustness, even for complex representations. They are effective in cluttered environments, low-lighting scenarios, and occlusion. Their limitations are that they generally have larger computational requirements and must be trained on large annotated datasets, and the quality of the training data directly affects tracking performance. These methods use deep neural networks to directly estimate objects' pose from the input image. YOLO-6D [25] is a well-known method in this category. It uses a convolutional neural network architecture to match and predict an object pose from a single image in real time. Furthermore, [26] can be given as a state-of-the-art example in this category. The authors propose a novel Deep Fusion Transformer Block, which proves to be very robust under varying environmental conditions.

The integration of deep learning techniques with traditional methods for pose estimation has shown significant advantages. Deep learning algorithms, particularly convolutional neural networks (CNNs), excel at extracting high-level features from images, enhancing pose estimation's robustness and accuracy. When combined with traditional methods like geometric approaches or optimization-based techniques, these algorithms can leverage the strengths of both worlds. Traditional methods contribute well-understood, understandable models and fast computation, while deep learning provides superior feature extraction and adaptability. RNNPose [27] is an example of such a system. The authors combine a recurrent neural network with a Levenberg-Marquardt optimization algorithm to obtain object pose.

2.2.2 Visual-Inertial Pose Tracking

Inertial measurement units (IMUs) have become widely available and cheaper over the years. They can provide motion data sampled at high frequencies and can be easily integrated into other objects or devices due to their compact size. Their measurements can be used to obtain accurate position and orientation estimates of the objects for short periods of time, but these estimates suffer from error accumulation over longer times [28].

IMU pose estimates can provide frequent updates with low latency and are unaffected by environmental conditions. This is why inertial measurements are often used with other less frequently updated sensors like cameras that provide non-drifting measurements. This way, the tracking system benefits from the advantages of both modalities.

Depending on the application, several setups and algorithms are used in object pose tracking with IMU and cameras. One very common setup is mounting a camera and IMU on a single platform and tracking the platform's 6 DoFs. This is also referred to as inside-out tracking. This setup can be seen in AR, VR, and robotics applications [29, 30, 31, 32]. The advantages of these systems are that they do not need an external setup of cameras, and they reach good accuracy. For instance, [32] reports sub 1° mean accuracy for orientation tracking and sub 8 mm accuracy for positional tracking. Another way of tracking the pose of objects is a setup with a fixed camera and a moving object. This tracking method is called outside-in tracking [33]. In such systems, the object must constantly remain in the camera's field of view (FoV) and, therefore, have a more restricted application field. Nevertheless, they require less specialized hardware, and they benefit from fixed reference points, which can increase their accuracy under different environmental conditions.

Another important aspect of visual-inertial tracking is the utilized data fusion algorithms. The most common fusion algorithms are the Kalman filter-based algorithms [31, 32]. These algorithms are computationally efficient and perform well with well-understood models. Another contender is particle filter-based methods, which allow flexibility for non-linear models, making them suitable for complex, dynamic environments [34]. Some downsides are that they are computationally expensive and less scalable. Finally, in recent years, some deep-learning-based methods have been proposed that offer great accuracy but generally require large training datasets and are also computationally demanding [35].

Chapter 3

Theory

This chapter provides a theoretical background on the core principles behind object tracking, focusing on both camera-based and IMU-based tracking techniques. First, object tracking concepts, such as the dynamics and coordinate systems essential for describing object motion, are introduced, followed by an overview of orientation representation methods. The chapter continues with a discussion of tracking techniques, including the use of inertial measurement units and fiducial markers for camera tracking. Finally, the chapter concludes with an explanation of the novel sensor fusion algorithm developed in this thesis.

3.1 Object Tracking

Object tracking is essential in various fields, including navigation, autonomous driving, and, more recently, the augmented and virtual reality (AR-VR) space. Although being used in different fields, the fundamentals of object tracking remain the same. At its core, object tracking refers to determining the pose of an object, which includes its position and orientation in a predetermined coordinate system over time. As discussed in chapter 1, object tracking plays a crucial role in model-mediated bilateral teleoperation. To maintain the accuracy and validity of the local model, the objects in the remote environment must be tracked. Ideally, this tracking occurs with minimal latency as the model parameters must be transmitted over the network to the operator domain. One of the main challenges of bilateral teleoperation is to keep the model as accurate as possible so that the operator can maintain hand-eye coordination.

In this thesis, we focus on tracking the pose of a single object in an empty environment and explore two main modalities for object tracking: camera tracking and inertial measurement unit (IMU) tracking. We analyze the strengths and weaknesses of both approaches and then combine them to develop a tracking method that leverages the advantages of each. A fundamental algorithm that is used for IMU tracking is the Kalman filter [36], a widely used optimal algorithm for the state estimation of linear systems. To address the non-linear dynamics of orientation estimation and facilitate the integration of IMU and camera tracking, we utilize a modified version of this filter known as the extended Kalman filter.

The pose of an object is typically referenced within a predetermined coordinate system. Position is generally expressed in x, y, z Cartesian coordinates, while orientation can be represented in various ways, each with its own set of advantages and limitations. Given the complexity of managing multiple coordinate systems or reference frames during object tracking, it is crucial to establish clear conventions for both the coordinate systems and orientation representations to ensure consistency and minimize confusion. In this thesis, the quaternion representation is predominantly used for orientation due to its computational efficiency and ability to avoid mathematical issues associated with other representations. The specific conventions that are used for the coordinate system and orientation are de-

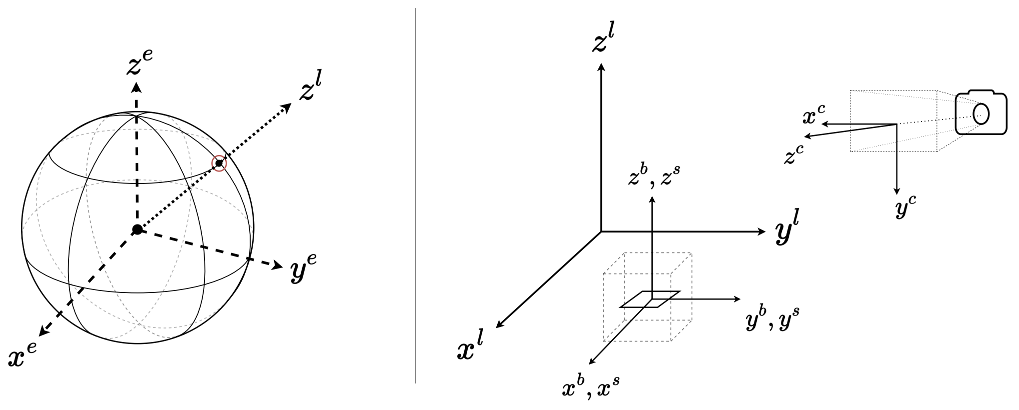


Figure 3.1: **Illustrations of the various coordinate systems used in this thesis. The earth frame, depicted on the left, has its axes labeled with the superscript e. On the right, the local, body, sensor, and camera frames are shown, with their respective axis superscripts l, b, s, and c.**

tailed further in subsection 3.1.1 and subsection 3.1.2.

3.1.1 Coordinate Systems

In this report, we need to establish several coordinate frames to discuss the dynamics of an object. We will follow the guidelines from [28]. The description of an object's movement depends on the coordinate system used, though the actual movement remains unchanged. The used coordinate systems are illustrated in Figure 3.1. First, the **earth frame**, ***e*** is assumed to lie on the center of the Earth's core and rotates with the Earth. Next, the **local frame**, ***l*** is introduced as the frame relative to which the object is tracked. The z axis of this frame is always assumed to originate from the earth's center and point outwards. This frame remains stationary with respect to the earth frame, providing a localized reference for the movement. The **body frame**, ***b*** is established directly at the object's center and is fixed to the object of interest. The **camera frame**, ***c*** represents another coordinate system that is stationary with respect to the local frame, allowing any pose to be transformed using a fixed transformation from the camera frame to the local frame. The center of this coordinate system is assumed to be at the camera's focal point, primarily used in camera tracking. Finally, the **sensor frame**, ***s*** is placed at the center of the IMU, aligned with the accelerometer and gyroscope axes. In the application, the sensor is fixed to the object. Therefore, the sensor frame is stationary with respect to the body frame, and poses can be transformed between the two frames using a fixed transformation. Later on in this section, this transformation is denoted as R_{imu}. In Figure 3.1 and the rest of this section, the body and sensor frames are assumed to be aligned.

3.1.2 Orientation Representation

The orientation of an object can be represented in various ways, each offering different properties. One widely recognized form is the Euler angle representation, which consists of a 3D vector of roll, pitch, and yaw angles. These angles simply correspond to rotations about the object's intrinsic axes. While Euler angles provide an intuitive understanding of orientation, they have limitations. One significant issue is *Gimbal Lock*, a situation where two or three axes become aligned, causing a loss of one degree of freedom. This restricts the use of Euler angles in certain situations. Additionally, calculations with Euler angles involve trigonometric functions, which can increase computational costs and complexity. Despite these drawbacks, Euler angles are still used to describe certain rotations in this thesis. However, to overcome these limitations, quaternion representation is predominantly employed to represent orientation.

Quaternions for Orientation Representation Quaternions are complex numbers made up of a 4D vector that includes a scalar real part (λ) and a 3D imaginary vector component ($\boldsymbol{\phi}$) as in

$$\boldsymbol{q} = [\lambda, \boldsymbol{\phi}^T]^T = [q_0, q_1, q_2, q_3]^T. \tag{3.1}$$

A quaternion whose norm is 1 is known as a unit quaternion. Unit quaternions are particularly useful for representing rotations in 3D space, as they avoid singularities and discontinuities present in other methods. The axis-angle representation is used to describe a rotation using a unit quaternion. If a rotation of angle θ around a unit vector axis $\hat{\boldsymbol{n}}$ is to be represented, the components λ and $\boldsymbol{\phi}$ of the corresponding unit quaternion $\boldsymbol{q}$ are given by:

$$\lambda = \cos\left(\frac{\theta}{2}\right), \quad \boldsymbol{\phi} = \sin\left(\frac{\theta}{2}\right)\hat{\boldsymbol{n}}. \tag{3.2}$$

Here, λ corresponds to the cosine of half the rotation angle, and $\boldsymbol{\phi}$ is the product of the sine of half the angle and the unit vector $\hat{\boldsymbol{n}}$ that indicates the axis of rotation.

One of the key features of quaternions is their ability to represent rotations through multiplication. Given two quaternions $\boldsymbol{q_1}$ and $\boldsymbol{q_2}$, their product $\boldsymbol{q}'$ is another quaternion representing the combined rotation:

$$\boldsymbol{q}' = \boldsymbol{q_1}\boldsymbol{q_2}. \tag{3.3}$$

The order in which the quaternions are multiplied matters and affects which orientation the combined quaternion represents. The quaternion multiplication is defined for two quaternions $\boldsymbol{p} = [p_0, \boldsymbol{p}_v]^T$, and $\boldsymbol{q} = [q_0, \boldsymbol{q}_v]^T$ in [28] as follows:

$$\boldsymbol{p}\boldsymbol{q} = \begin{bmatrix} p_0 q_0 - \boldsymbol{p}_v \cdot \boldsymbol{q}_v \\ p_0 \boldsymbol{q}_v + q_0 \boldsymbol{p}_v + \boldsymbol{p}_v \times \boldsymbol{q}_v \end{bmatrix}. \tag{3.4}$$

To rotate a vector $\boldsymbol{v}$ using a quaternion $\boldsymbol{q}$, the vector is first converted into a quaternion $\boldsymbol{v}_q$ with 0 as the scalar part: $\boldsymbol{v}_q = [0, v_x, v_y, v_z]$. The rotated vector $\boldsymbol{v}'_q$ is obtained as the following:

$$\boldsymbol{v}'_q = \boldsymbol{q}\boldsymbol{v}_q\boldsymbol{q}*, \tag{3.5}$$

where $\boldsymbol{q}*$ is the conjugate of $\boldsymbol{q}$ defined as

$$\boldsymbol{q}* = [q_0, -q_1, -q_2, -q_3]. \tag{3.6}$$

Rotation Matrices Another common way to apply a rotation to a vector in Euclidean space is through a rotation matrix $R \in \mathbb{R}^{3\text{x}3}$. For a vector $\boldsymbol{p} \in \mathbb{R}^3$, the rotated vector $\boldsymbol{p}'$ can be obtained by:

$$\boldsymbol{p}' = R\boldsymbol{p}. \tag{3.7}$$

A unit quaternion can be converted into a rotation matrix using the following formula [37], which provides a convenient way to apply the quaternion's rotation to any vector:

$$R = I_3 + 2\lambda[\boldsymbol{\phi}]_\times + [\boldsymbol{\phi}]_\times^2, \tag{3.8}$$

where I_3 is the 3×3 identity matrix, and the $[.]_\times$ operator represents the skew-symmetric matrix of its operand, defined for a vector $\boldsymbol{v} = (v_x, v_y, v_z)^T$ as:

$$[\boldsymbol{v}]_\times = \begin{bmatrix} 0 & -v_z & v_y \\ v_z & 0 & -v_x \\ -v_y & v_x & 0 \end{bmatrix}. \tag{3.9}$$

3.2 Object Tracking with an IMU

A notable approach that has gained widespread acceptance in the industry for various applications is tracking an object with an IMU. The working principle behind IMUs allows for tracking objects without relying on specific object properties. The compact size of MEMS (Micro-Electro-Mechanical Systems) IMUs makes them particularly suitable for embedding in objects of any size. Additionally, IMUs are often used to complement other object-tracking methods due to their various advantages. In this section, we discuss the pros and cons of using only IMUs for tracking objects. We also propose models for the sensors we utilize and describe their application within the tracking algorithm. This analysis will be used later to compare with the proposed object-tracking method in this thesis.

3.2.1 Inertial Measurement Unit

IMUs are typically equipped with an accelerometer and a gyroscope. The accelerometer measures the acceleration on the sensor along three orthogonal axes, while the gyroscope measures the rotational velocity of the sensor around these axes. These measurements can be used to calculate the sensor's pose starting from an initial reference point. Double integration of the acceleration data provides the displacement of the device, while single integration of the rotational velocity yields the change in the device's orientation.

A significant advantage of using IMUs is their ability to provide high-frequency data, often reaching multiple kilohertz (kHz), with minimal processing required for pose estimation. This capability is useful for detecting sudden changes during tracking and reduces the latency between movement and pose estimation. However, a notable drawback of using IMUs for tracking is the presence of significant noise components in the measurements. Because the calculations rely on integrating these measurements and are relative to an initial pose, substantial drift can accumulate over time.

One effective method to mitigate drift is using estimation filters, such as the Kalman filter, which is primarily employed in this thesis. The application of the Kalman filter will be discussed in subsequent sections. Additionally, it is crucial to calibrate the IMU sensors before use, as proper calibration can significantly delay the onset of drift in the estimations.

Accelerometer As mentioned, the accelerometer measures the linear acceleration acting on the sensor. In addition to this, it detects a constant gravity vector that points outward from the center of the earth frame and is aligned with the z axis of the local frame. This means that when laying flat on a surface, the accelerometer measures a constant acceleration of 1 G (9.806 m/s^2) and registers 0 G during free fall. However, the accuracy of the accelerometer measurements is affected by several error sources, including measurement noise, scaling errors, bias, and axis misalignment. To accurately utilize accelerometer data for object tracking algorithms, it is essential to adopt a mathematical model that accounts for these errors. This thesis uses the mathematical model for the accelerometer as derived in [32], represented by the following:

$$\boldsymbol{a}_A = R_{\text{IMU}} R^T(\boldsymbol{q})\big(\boldsymbol{a} + \boldsymbol{g} + H(\boldsymbol{\omega}, \boldsymbol{a}) O_{\text{IMU}}\big) + \boldsymbol{b}_A + \boldsymbol{\delta}_A. \tag{3.10}$$

In this model, R_{IMU} is the rotation matrix from the sensor to the body frame, $\boldsymbol{q}$ denotes the rotation quaternion of the sensor frame relative to the local frame, with $R(.)$ converting the quaternion into a rotation matrix using Equation 3.8. The term $\boldsymbol{a}$ signifies the sensor's ideal linear acceleration, $\boldsymbol{g}$ stands for gravitational acceleration, and $H(\boldsymbol{\omega}, \boldsymbol{a}) O_{\text{IMU}}$ encompasses the effects of Coriolis forces on the acceleration measurements. In [32], $H(\boldsymbol{\omega}, \boldsymbol{\alpha}) = [\boldsymbol{\omega}]_\times^2 + [\boldsymbol{\alpha}]_\times$ and O_{IMU} is given as the offset of the sensor frame with respect to the body frame of the object. In this thesis, since these two frames are assumed to coincide, the effects of Coriolis forces on the model are neglected. Furthermore, $\boldsymbol{b}_A$ represents the accelerometer bias which is assumed to be a constant offset, while $\boldsymbol{\delta}_A$ denotes the measurement noise, which is assumed to follow a zero-mean Gaussian distribution. This comprehensive model allows for a more accurate interpretation of accelerometer data by accounting for various errors and forces that may affect the measurements.

Gyroscope The gyroscope measures the angular velocity relative to the sensor frame, which is crucial in determining the rotational movements of the device. Accurate orientation data obtained through the gyroscope is also essential for compensating for the influence of the gravity term $\boldsymbol{g}$, as highlighted by the accelerometer's mathematical model. However, gyroscope readings are subject to similar error sources as the accelerometer, including noise and bias. The following mathematical model, also derived in [32], is used to account for these factors:

$$\boldsymbol{\omega_G} = R_{\text{IMU}}\boldsymbol{\omega} + \boldsymbol{b}_G + \boldsymbol{\delta}_G. \tag{3.11}$$

Within this model, $\boldsymbol{\omega}_G$ signifies the gyroscope's measured angular velocity, while R_{IMU} represents the constant rotation matrix of the sensor frame relative to the body frame. The term $\boldsymbol{\omega}$ denotes the true angular velocity. Additionally, $\boldsymbol{b}_G$ refers to the constant gyroscope bias, and $\boldsymbol{\delta}_G$ accounts for the measurement noise, which is modeled as a zero-mean Gaussian distribution.

3.2.2 Kalman Filtering

The Kalman filter, introduced by [36], is a widely used optimal state estimation filter. It estimates the state of a linear system by utilizing a mathematical model and noisy measurements of the system state. The filter assumes that the system it models is linear and that the noise within the system follows a zero-mean Gaussian distribution. This makes the Kalman filter particularly suitable for estimating the position of an object using accelerometer data, as the translational component of the pose estimation aligns with the filter's linear system model requirements.

The Kalman filter operates in two main steps: a time update (prediction) step and a measurement update (correction) step. The time update forecasts the system's next state using the system model, as outlined by the following equation:

$$\boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k) + \boldsymbol{\omega}_k. \tag{3.12}$$

In this equation, the system function f predicts the next state $\boldsymbol{x}_{k+1}$ by using the current state estimate $\boldsymbol{x}_k$, the control input vector $\boldsymbol{u}_k$, and an additive noise component $\boldsymbol{w}_k$. The noise component represents the uncertainties in the system's dynamics. Furthermore, a time component is implicitly included in the system function, generally as a small time step. In the measurement update step, the prediction is refined by incorporating actual measurements of some state variables using an observation model:

$$\boldsymbol{z}_k = h(\boldsymbol{x}_k) + \boldsymbol{\epsilon}_k. \tag{3.13}$$

Here, h represents the observation model that maps the predicted state $\boldsymbol{x}_k$ to the measurement $\boldsymbol{z}_k$. The term $\boldsymbol{\epsilon}_k$ denotes the measurement noise, which is also assumed to follow a zero-mean Gaussian distribution. By utilizing these models and iterating through a series of steps, the Kalman filter continuously updates its estimates of the system's state, effectively reducing the uncertainty and noise inherent in the measurements. This process results in a more accurate estimation of the system's true state after each measurement of the states.

The Kalman filter not only estimates the state of the system but also keeps track of the uncertainties associated with these state estimates, updating them with each prediction and update step. This tracking of uncertainties is crucial, as it determines how much weight should be assigned to the predicted state and the measured state during the update process. The weight, dynamically calculated at each step, is known as the Kalman gain. Table 3.1 shows the individual steps that are performed during both steps.

Time Update	**Measurement Update**
1: $\hat{\boldsymbol{x}}_k^- = F\hat{\boldsymbol{x}}_{k-1} + G\boldsymbol{u}_k$	3: $K_k = \hat{P}_k^- H^T(H\hat{P}_k^- H^T + R)^{-1}$
2: $\hat{P}_k^- = F\hat{P}_{k-1}F^T + Q$	4: $\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + K_k(z_k - H\hat{\boldsymbol{x}}_k^-)$
	5: $\hat{P}_k = (I - K_k H)\hat{P}_k^-$

Table 3.1: **Linear Kalman Filter Equations [32]**

Tracking position with a linear Kalman filter

As mentioned before, the position of the object can be tracked using the accelerometer measurements and a Kalman filter. This can only be done accurately if the constant gravity vector of the accelerometer is compensated, usually by estimating the orientation of the sensor first. In the following explanation, the components of the gravity vector are assumed to be subtracted from the acceleration measurements.

Each time new measurements for the three axes are obtained, the steps detailed in Table 3.1 are performed. In the first line of Table 3.1, $\hat{\boldsymbol{x}}_{k-1}$ denotes the current state vector before state estimation. The hat symbol indicates an estimate. The state vector for position estimation can be constructed as in Equation 3.14:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} & \boldsymbol{v} & \boldsymbol{a} \end{bmatrix}^T . \tag{3.14}$$

Here $\boldsymbol{p}$ denotes the position vector, $\boldsymbol{v}$ denotes the linear velocity vector, and $\boldsymbol{a}$ denotes the linear acceleration vector (not including a gravity component). At the very first run of the Kalman filter, the value of the state vector needs to be initialized. For this system, these values can all be initialized to start from 0 as we can determine that the origin of the local frame coincides with the body frame, and the object begins in a stationary position. Other systems where the initial state is unknown should be initialized with either an initial measurement or an educated guess.

The F matrix in line 1 is the state transition matrix and models the system dynamics. The F matrix for the position tracking system is as follows:

$$F = \begin{bmatrix} 1 & 0 & 0 & \Delta T & 0 & 0 & \frac{1}{2}\Delta T^2 & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta T & 0 & 0 & \frac{1}{2}\Delta T^2 & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta T & 0 & 0 & \frac{1}{2}\Delta T^2 \\ 0 & 0 & 0 & 1 & 0 & 0 & \Delta T & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta T & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & \Delta T \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} . \tag{3.15}$$

From Equation 3.15, it can be seen that the position of the system is obtained by integrating the velocity, and the velocity is obtained by integrating the acceleration. The acceleration is predicted to be stationary in this model. ΔT refers to the time between the current and previous prediction steps. The G matrix in line 1 is called the control matrix, and $\boldsymbol{u}$ is the input vector. Since there are no external inputs in our system, the control matrix can be considered empty.

In the second line of Table 3.1, The error covariance matrix P is predicted into the future. The error covariance matrix is the matrix that keeps track of the uncertainties associated with each state. The diagonal elements in this matrix indicate the variance of the state variables, while the off-diagonal elements represent the covariance between all the state variables. At the initial run, this matrix also needs to be initialized. The initialization for our system can be done by setting all diagonal entries to 0.001 while setting all off-diagonal entries to 0. Usually, a very low initial variance is set, indicating that the initial guess for the state is very accurate. Matrix Q is the process noise covariance matrix. This matrix has the same dimensions as the process error covariance matrix. This matrix represents the uncertainty in the dynamics of the system. This matrix is needed since the model dynamics will contain randomness that cannot be represented in the model itself.

In the third line of Table 3.1, the Kalman gain K_k is calculated. The Kalman gain is used as a way to determine which component of the system should be given more weight, either the measurement or the outcome of the model dynamics equations. The H matrix is the observation matrix that relates the state vector to the obtained measurements. The H matrix for the position tracking system can be seen in Equation 3.16:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.16}$$

Only the accelerations are measured. The R matrix is the measurement noise covariance matrix. Just like the process noise covariance matrix, this matrix captures the uncertainties in the measurements. This matrix should be set with the specified or measured variances of each sensor that is used in the system. In the 4^{th} and 5^{th} lines of Table 3.1, the state vector and the error covariance matrix are updated with the help of the obtained measurements. $\boldsymbol{z}_k$ is the measurement vector given as follows

$$\boldsymbol{z} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T. \tag{3.17}$$

With these equations, the new state estimate $\hat{\boldsymbol{x}}_k$ is calculated alongside the new state uncertainties in the error covariance matrix $\hat{P}_k$. It must be noted that in the setting of this linear Kalman filter, there is no mitigation of the constant gravity vector being measured by the accelerometer. As long as the orientation of the object does not change, this gravity vector can simply be removed from the measurement obtained on the sensor's z-axis. Nevertheless, in more complex situations where the object orientation also changes, the exact orientation of the object must be known in order to remove this component. This is explained in the following section.

Initialization and tuning At the startup of the Kalman filter, it is essential to initialize both the state vector and the error covariance matrix. Assuming the object is stationary at the beginning, the state vector can be initialized to zero. The error covariance matrix, on the other hand, should be initialized based on the confidence in this initial state estimate: high values in the error covariance matrix indicate low confidence and assume that the initial state may be significantly erroneous, while smaller values reflect high confidence. Another important consideration is the initialization of the process noise covariance matrix Q and the measurement noise covariance matrix R. Setting R is relatively straightforward, as it can be derived from the variance of the sensor measurements. In contrast, determining the appropriate Q is more challenging due to the complexities in accurately modeling the process noise. This matrix can be initialized by tuning the system with trial and error.

Tracking orientation with an extended Kalman filter

As discussed in the previous chapter, knowing the orientation of the sensor is crucial for accurate position estimation and for correctly subtracting the gravity vector from acceleration measurements. However, directly applying the linear Kalman filter in combination with gyroscope measurements to estimate orientation is not feasible because orientation dynamics are inherently non-linear. Various algorithms exist for calculating orientation, such as the Madgwick Filter [38], which is a type of complementary filter. While algorithms like the Madgwick Filter offer computational efficiency, the EKF provides a more robust framework for handling the complex non-linearities associated with orientation estimation in our specific application. In this thesis, we will use the extended Kalman filter (EKF), an advanced version of the Kalman filter that is designed to handle non-linear systems. The EKF extends the applicability of the linear Kalman filter by linearizing the non-linear system around the current estimate, allowing it to be used for a broader class of non-linear dynamic systems. The extended Kalman filter has the same steps as the linear Kalman filter, with slight changes in the equations. These equations can be seen in Table 3.2.

Time Update	**Measurement Update**
1: $\hat{\boldsymbol{x}}_k^- = f(\hat{\boldsymbol{x}}_{k-1}, \boldsymbol{u}_k, \boldsymbol{\omega}_k)$	3: $K_k = \hat{P}_k^- H_k^T (H_k \hat{P}_k^- H_k^T + R)^{-1}$
2: $\hat{P}_k^- = F_k \hat{P}_{k-1} F_k^T + Q$	4: $\hat{\boldsymbol{x}}_k = \hat{\boldsymbol{x}}_k^- + K_k(z_k - h(\hat{\boldsymbol{x}}_k^-))$
	5: $\hat{P}_k = (I - K_k H_k)\hat{P}_k^-$

Table 3.2: **Extended Kalman Filter Equations [32]**

For the orientation of the system, we need to establish the state vector first. As discussed earlier, we can represent the orientation of an object using quaternions so the state vector can be constructed as follows:

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q}^T & \boldsymbol{w}^T \end{bmatrix}^T, \tag{3.18}$$

where $\boldsymbol{q}$ is the unit quaternion representing the orientation of the object's body frame with respect to the local frame, and $\boldsymbol{w}$ are the angular velocities of the object in three axes. The angular velocities are included as this is the physical quantity we can actually measure with the gyroscope. From Table 3.2, it can be seen that the first line has changed relative to Table 3.1. Since we can't linearly relate the angular velocities in the state vector to orientation, we need to calculate the next system state using a non-linear function, which is described in the first line. The non-linear system dynamics function of our system is as follows:

$$\hat{\boldsymbol{x}}_{\boldsymbol{k}}^- = \begin{bmatrix} \boldsymbol{q}_{k-1} + \Delta T \dot{\boldsymbol{q}}_{k-1} \\ w_{k-1} \end{bmatrix}, \tag{3.19}$$

where,

$$\dot{\boldsymbol{q}}_{k-1} = \frac{1}{2}\boldsymbol{q}_{k-1} \otimes w_{k-1}^T. \tag{3.20}$$

With the upper equation in Equation 3.19, the change in each quaternion component in a single time step is obtained by multiplying a small time step with the derivative of the quaternion at the previous state. A quaternion derivative can be obtained with the angular velocity according to Equation 3.20, assuming the time step is very small. This is done by applying a quaternion multiplication between the current state quaternion and the current state angular velocities in quaternion representation. This representation is obtained by making a 4D quaternion-like vector from the angular velocities where the scalar component is set to 0. The angular velocities are assumed to be stationary in this dynamic system model. Another change between the two equation tables is in line 4, where the observation model is used. The observation model, as discussed in subsection 3.2.1 for the gyroscope can be given as,

$$h(\boldsymbol{x}_k) = [R_{\text{IMU}} \boldsymbol{w}_k + \boldsymbol{b}_{G,k}]. \tag{3.21}$$

Since we assumed the sensor frame to coincide with the body frame, the term R_{IMU} denotes no rotation and the observation model reduces to

$$h(\boldsymbol{x}_k) = [\boldsymbol{w}_k + \boldsymbol{b}_{G,k}]. \tag{3.22}$$

Since in the extended Kalman filter, a non-linear dynamic system model and a non-linear observation model are used, the F and H matrices are obtained by linearizing these models at the current estimate. Specifically, F is the Jacobian of the state transition function, and H is the Jacobian of the observation function, both with respect to the state vector. These Jacobian matrices are evaluated at the current state estimate, effectively linearizing the non-linear models around the current point of estimation at each time step. The Jacobian matrix is constructed by taking the partial derivatives of each component of the function with respect to each element of the state vector, thereby forming a matrix of these partial derivatives. The Jacobian F is obtained by deriving Equation 3.19 with respect to Equation 3.18 as in

$$F = \left.\frac{\partial f(x,u)}{\partial x}\right|_{x=\hat{x_k}}. \tag{3.23}$$

For the orientation estimation system, this will result in :

$$F_k = 0.5 \begin{bmatrix} 2 & -\Delta T w_x & -\Delta T w_y & -\Delta T w_z & -\Delta T q_1 & -\Delta T q_2 & -\Delta T q_3 \\ \Delta T w_x & 2 & \Delta T w_z & -\Delta T w_y & \Delta T q_0 & -\Delta T q_3 & \Delta T q_2 \\ \Delta T w_y & -\Delta T w_z & 2 & \Delta T w_x & \Delta T q_3 & \Delta T q_0 & -\Delta T q_1 \\ \Delta T w_z & \Delta T w_y & -\Delta T w_x & 2 & -\Delta T q_2 & \Delta T q_1 & \Delta T q_0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \end{bmatrix}. \tag{3.24}$$

Similarly, to obtain the Jacobian H the following equation is constructed with Equation 3.22 and Equation 3.18:

$$H = \left.\frac{\partial h(x)}{\partial x}\right|_{x=\hat{x_k}}. \tag{3.25}$$

For the orientation estimation system, this will result in:

$$H = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.26}$$

With these Jacobians and the non-linear system dynamics and observation models, the equation steps in Table 3.2 can be applied with the gyroscope measurements to estimate the orientation of the object. An important thing to note is that after every time update and measurement update step, the obtained new quaternions will not be unit quaternions anymore. The newly estimated quaternions need to be normalized according to

$$\boldsymbol{q}_{unit} = \frac{\boldsymbol{q}}{||\boldsymbol{q}||} = \left[\frac{q_0}{||\boldsymbol{q}||}, \frac{q_1}{||\boldsymbol{q}||}, \frac{q_2}{||\boldsymbol{q}||}, \frac{q_3}{||\boldsymbol{q}||}\right], \tag{3.27}$$

where,

$$||\boldsymbol{q}|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}. \tag{3.28}$$

3.3 Object Tracking with a Camera

Camera-based object tracking is a well-studied field and forms an important part of computer vision. To track an object with a camera, one or multiple specific features of the object must be known, which can then be used to track the object's pose in the obtained camera frames. When a common RGB camera is used for object tracking, these features typically consist of easily recognizable patterns such as edges, corners, colors, blobs, and distinct shapes.

A widely used approach for this is machine learning or pattern recognition, where systems can be trained with example features so that they can recognize these features in the frames. Machine learning can be a powerful tool for tracking objects when the nature of the objects is not known beforehand, and it scales well for multiple object tracking. Nevertheless, these methods are known to require large amounts of processing power and, in some cases, have long training times.

In cases where the object of interest is known and available for manipulation before tracking starts, a more convenient and reliable way to make object features known to the system is by embedding predefined markers on the object. This approach eliminates the need for extensive training and allows for relatively easy setup. The processing time is significantly improved because the system knows exactly what features to look for. Since this is also the case in this thesis, a marker-based object-tracking method is utilized and is explained in further detail in this section.

While marker-based methods improve certain aspects, it is important to note that they are still susceptible to inherent limitations of camera tracking. The accuracy and reliability of camera tracking can suffer from several factors, such as occlusion, insufficient lighting, background clutter, motion blur, noise, incorrect depth estimations, complexity, and false positives. Furthermore, there is a significant processing delay when compared to other modalities, such as the IMU method discussed before.

3.3.1 Tracking with Fiducial Markers

In marker-based tracking, a predefined set of markers is placed on an object, with the relative pose of these markers to the object being known and constant. By recording the object using a stationary RGB camera, the poses of these markers relative to the camera can be estimated using a 2D-to-3D transformation method, such as the Perspective-n-Point (PnP) algorithm [19]. In this transformation, the 2D coordinates obtained from the camera frames are matched with the known 3D positions of the marker points. Since the relative transformation of the markers to the object is known and static, the object's pose can be accurately estimated by combining the marker pose estimates from each frame, thus enabling continuous tracking of the object.

ArUco markers and marker detection

ArUco markers are rectangular markers featuring unique bit patterns at their center. An example of a marker consisting of a 6 by 6 pattern can be seen in Figure 3.2. These bit patterns encode a unique binary code that identifies different markers within the same image and includes error detection and correction mechanisms [39]. The four corners of each marker are utilized to estimate its pose relative to the camera.

ArUco markers are designed to maximize the inter-marker distance within a predefined set of markers or a dictionary to minimize false positives and incorrect marker identification. This feature facilitates robust pose estimation when multiple markers are used on the same object. Moreover, the built-in error detection and correction are particularly advantageous in challenging conditions such as low camera resolution, rapid camera movements, poor lighting, and occlusion.

The process of detecting and identifying ArUco markers involves several steps:

Image Preprocessing The acquired RGB image is converted to grayscale and then transformed into a binary image using a local adaptive threshold. This simplifies the image and highlights the markers against the background.

Contour Extraction A contour extraction algorithm [40] is employed to identify the outlines of polygons within the binary image.

Contour Filtering Filters are applied to the detected contours to eliminate shapes that do not meet the criteria of potential markers—specifically, non-rectangular shapes, concave polygons, inner contours, and other incompatible forms.

Marker Binary Extraction For each candidate marker, a homography matrix is computed to correct perspective distortion, effectively "flattening" the marker. The corrected image is divided into a square grid, and Otsu's thresholding method [41] is applied to binarize each cell. The binary code that represents the unique ID of the marker is then extracted.

By completing these steps, the corner points and unique IDs of the markers present in the image are obtained. These corner points are crucial for estimating the pose of each marker, which in turn enables precise tracking of the object's position and orientation in space.

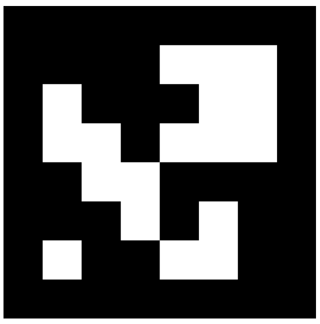


Figure 3.2: **A typical example of a 6x6 AruCo marker. The marker can be used to obtain the pose of an object on which it is placed using a camera tracking algorithm.**

3.3.2 Marker Pose Estimation

The marker's pose can be calculated after extracting the set of corner points from the image. The objective is to compute the pose of the marker in the coordinate system of the stationary camera, a problem commonly referred to as the Perspective-n-Point (PnP) problem. To solve this problem, we utilize the pinhole camera model, as illustrated in Figure 3.3. This model provides a mathematical framework for understanding how a 3D point in the world coordinate system projects onto a 2D point in the image plane. It assumes the camera is a point in space without lenses, simplifying the relationship between the coordinates.

According to [42], the relationship between the world coordinates and the image plane coordinates using the pinhole camera model is expressed as:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = P \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} \tag{3.29}$$

In this equation, u, v, and w are the homogeneous coordinates on the image plane. The actual image coordinates are obtained by normalizing these values: $x = u/w$ and $y = v/w$. The vector $[X_w, Y_w, Z_w]^T$ represents the homogeneous coordinates of a point in the world coordinate system. The matrix P is the camera projection matrix. The camera projection matrix P can be decomposed into the intrinsic matrix K and the extrinsic matrices $[R|T]$, as shown:

$$P = K\ [R \mid T]. \tag{3.30}$$

The intrinsic matrix K contains the internal parameters of the camera, such as the focal lengths f_x and f_y, and the principal point coordinates c_x and c_y:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.31}$$

The principle point is the point where the z axis of the camera frame meets the image plane. These parameters are specific to the camera and are obtained through a calibration process. The calibration process for these parameters is explained later in this section.

The extrinsic matrix $[R|T]$ consists of the rotation matrix R and the translation vector T:

$$[R|T] = \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_x \\ R_{21} & R_{22} & R_{23} & T_y \\ R_{31} & R_{32} & R_{33} & T_z \end{bmatrix} \tag{3.32}$$

Here, R represents the rotation of the camera frame (or the camera itself) relative to the earth frame, and T represents the translation (position) of the camera in the earth frame. By substituting the intrinsic and extrinsic matrices back into the original equation, we relate the 3D world points to their 2D projections on the image plane. To solve for the marker's pose, we use the known 3D coordinates of the marker's corners (in the earth coordinate system) and their corresponding 2D projections (in the image plane). This forms a set of equations that can be solved using algorithms designed for the PnP problem.

One effective method for solving these equations is the Levenberg-Marquardt optimization algorithm [43], which iteratively refines the estimate of the rotation and translation vectors (called *rvec* and *tvec* in the OpenCV [44] framework) to minimize the reprojection error. The reprojection error is the difference between the observed 2D image points and the projected 3D points using the current estimate of the pose. The specifics of this algorithm are beyond the scope of this thesis. Once *rvec* and *tvec* are obtained, they represent the pose of the marker relative to the camera frame. Depending on the application, it may be necessary to invert these transformations to obtain the camera's pose relative to the marker or to express the pose in a different coordinate system.

3.3.3 Camera Calibration

While the objective in marker pose estimation is to find the extrinsic parameters of the camera, the intrinsic parameters must be known beforehand. Intrinsic parameters define the internal characteristics of the camera that affect how 3D points are projected onto the 2D image plane. As seen previously in Equation 3.31, these parameters include the focal length (f_x and f_y), the coordinates of the principal point (c_x and c_y), and lens distortion coefficients that account for imperfections in the camera lens. These parameters can be obtained by calibrating the camera before tracking. Since these parameters do not change over time (unless camera settings change), they must be calculated only once.

Calibration is essential since inaccuracies in these parameters can lead to significant errors in pose estimation. The calibration process involves capturing multiple images of a known calibration pattern, such as a checkerboard or a ChArUco board, from different

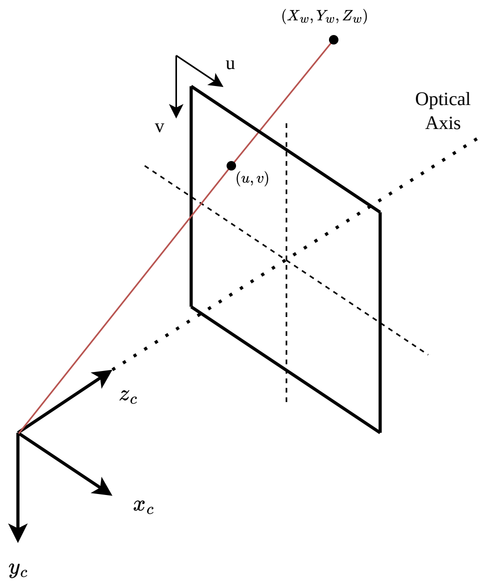


Figure 3.3: **Illustration of the pinhole camera model. 2D coordinates on the focal surface are represented by coordinates u, v, w. 3D world coordinates are represented by X_w, Y_w, Z_w. The length between the origin of the camera frame and the focal surface is the focal length of the camera and is camera-specific. The optical axis is aligned with the z_c axis of the camera frame.**

orientations and distances to cover as many different camera angles as possible. A ChArUco board is a board where the white tiles in a checkerboard are replaced with a series of ArUco markers to make the calibration process more robust. By analyzing the correspondences between the known 3D coordinates of the pattern points and their 2D projections in the images, calibration algorithms can solve for the intrinsic parameters. The OpenCV framework [44] incorporates many of these algorithms, which have also been used in this thesis.

3.4 Object Tracking using Sensor Fusion

As previously discussed, both inertial measurement unit (IMU) tracking and camera-based tracking systems have inherent advantages and disadvantages. IMU data offers low-latency tracking with frequent updates; however, it is prone to rapid drift due to the accumulation of sensor errors over short periods, which limits its accuracy. In contrast, camera-based tracking provides higher accuracy but delivers updates less frequently, and the required processing time introduces delays that make the estimates outdated upon arrival. In bilateral teleoperation settings, it is essential to obtain model parameters with minimal latency to directly update the local model, as any divergence can lead to significant synchronization issues. Considering these constraints, we have developed a novel object-tracking algorithm and system that exploits the strengths of both modalities. This algorithm is founded on the extended Kalman filter (EKF), and we have enhanced its capabilities by integrating additional steps beyond the conventional EKF procedure to improve performance and adaptability.

A high-level overview of the algorithm can be seen in Figure 3.4. The core of the algorithm consists of an EKF that uses the same EKF steps shown in Table 3.2. The aim

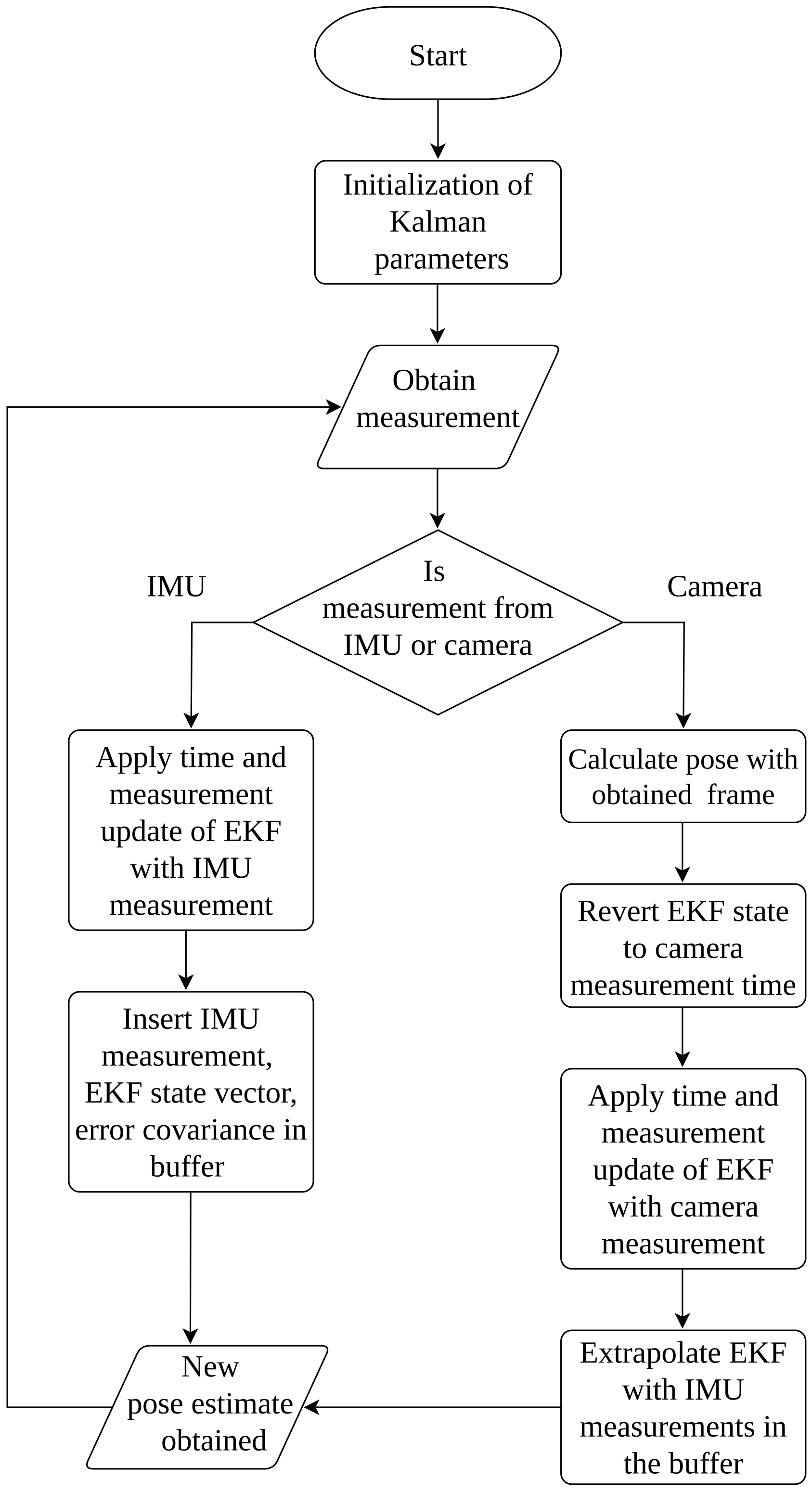

Start
Initialization of Kalman parameters
Obtain measurement
Is measurement from IMU or camera
IMU
Camera
Apply time and measurement update of EKF with IMU measurement
Insert IMU measurement, EKF state vector, error covariance in buffer
Calculate pose with obtained frame
Revert EKF state to camera measurement time
Apply time and measurement update of EKF with camera measurement
Extrapolate EKF with IMU measurements in the buffer
New pose estimate obtained

Figure 3.4: **Flowchart showing a high-level overview of the proposed sensor fusion algorithm.**

of this EKF is to fuse both IMU data and delayed camera pose estimates to get reliable high-frequency, low-latency pose estimates. To address the challenge of delayed camera measurements, the system implements a buffering mechanism that stores the last n states, estimates, and IMU measurements of the EKF. When a camera measurement becomes available, corresponding to a specific point in the past, the system reverts the EKF to the state at that exact timestamp. It then performs a measurement update using the camera estimate, effectively correcting the past state estimate. Subsequently, the system reapplies the saved IMU measurements from the buffer to propagate the corrected state forward to the current time. This approach allows the system to incorporate accurate past camera data into the current estimate, effectively mitigating drift while maintaining low-latency updates.

The state vector of the EKF, $\boldsymbol{x}$ is defined as

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{q} & \boldsymbol{\omega} & \boldsymbol{\alpha} & \boldsymbol{p} & \boldsymbol{v} & \boldsymbol{a} & \boldsymbol{b}_G & \boldsymbol{b}_A \end{bmatrix}^T. \tag{3.33}$$

$\boldsymbol{x}$ is a 25×1 vector. $\boldsymbol{q} = [q_w, q_x, q_y, q_z]$ is the unit quaternion representing the orientation of the object frame with respect to the local frame, $\boldsymbol{\omega}$ is the angular velocity of the object around the three principle axes of the body frame, $\boldsymbol{\alpha}$ is the angular acceleration in the three principle axes of the local frame, $\boldsymbol{p}$ is the position of the center of the body frame with respect to the origin of the local frame, $\boldsymbol{v}$ is the velocity of the object in the three principle axes of the local frame, $\boldsymbol{a}$ is the linear acceleration in the local frame, $\boldsymbol{b}_G$ are the gyroscope biases for each axis, and $\boldsymbol{b}_A$ are the accelerometer biases for each axis. By using this algorithm, the principle aim is to get accurate estimates for $\boldsymbol{q}$ and $\boldsymbol{p}$ to get the estimates for orientation and the position of the object.

The dynamics of the object are modeled with the process dynamics model as follows:

$$f(\boldsymbol{x}_k) = \boldsymbol{x}_k + \Delta T \begin{bmatrix} \dot{\boldsymbol{q}}_k & \boldsymbol{\alpha}_k & 0_3 & \boldsymbol{v}_k & \boldsymbol{a}_k & 0_9 \end{bmatrix}^T = \boldsymbol{x}_{k+1}^-. \tag{3.34}$$

In this equation, ΔT is the time step between the previous update and the current update. $\dot{\boldsymbol{q}}$ represents the derivative of the quaternion and can be calculated with Equation 3.20. The updated angular velocity is obtained by multiplying the time step with the angular acceleration $\boldsymbol{\alpha}$. The angular acceleration is assumed to stay constant during an update. The updated position $\boldsymbol{p}$ is the time step multiplied by the current linear velocity $\boldsymbol{v}$, and the updated linear velocity is the current linear acceleration $\boldsymbol{a}$ multiplied by the time step, and the linear acceleration itself is assumed to stay constant. Furthermore, in the model, the gyroscope and accelerometer biases, $\boldsymbol{b}_G$ and $\boldsymbol{b}_A$, are also modeled to be static. In the update step of the EKF, it is necessary to calculate the Jacobian matrix F of the system dynamics equation given in Equation 3.34. This matrix will yield the following result:

$$F = I_{25} + \Delta T \begin{bmatrix} diag\{\phi, \psi, 0_{6\times 6}\} \end{bmatrix}, \tag{3.35}$$

where,

$$\phi = \frac{1}{2} \begin{bmatrix} 0 & -\omega_x & -\omega_y & -\omega_z & -q_x & -q_y & -q_z \\ \omega_x & 0 & \omega_z & -\omega_y & q_w & -q_z & q_y \\ \omega_y & -\omega_z & 0 & \omega_x & q_z & q_w & -q_x \\ \omega_z & \omega_y & -\omega_x & 0 & -q_y & q_x & q_w \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{3.36}$$

and

$$\psi = \begin{bmatrix} 0_3 & I_3 & 0_3 \\ 0_3 & 0_3 & I_3 \\ 0_3 & 0_3 & 0_3 \end{bmatrix}. \tag{3.37}$$

Incorporating IMU measurements

The EKF uses two measurement modalities in the estimation process. The first sensor modality is the IMU embedded in the object, and the second is the pose estimation obtained with a marker pose estimation algorithm with a camera. As can be seen in Figure 3.4, the EKF operates based on which modality arrives at that time. The IMU data is assumed to arrive more frequently. Whenever IMU data arrives, a time update with the system dynamics function given in Equation 3.34 is applied, and afterward, a measurement update with the following observation model is applied:

$$h_{\text{IMU}}(\boldsymbol{x}_k) = \begin{bmatrix} R_{\text{IMU}}\boldsymbol{\omega}_k + \boldsymbol{b}_{G,k} \\ R_{\text{IMU}}R^T(\boldsymbol{q}_k)(\boldsymbol{a}_k + \boldsymbol{g}) + \boldsymbol{b}_{A,k} \end{bmatrix}. \tag{3.38}$$

In this model, the first row is the gyroscope model, also given in Equation 3.11, and the second row is the accelerometer model given in Equation 3.10. The difference in the accelerometer model is that since the accelerometer is assumed to be in the center of the object frame, the offset term O_{IMU} is 0, and therefore, the term that handles the Coriolis effect is omitted. In order to calculate the Kalman gain in the measurement update step of the EKF, the Jacobian H_{IMU} of the IMU measurement model needs to be calculated. This Jacobian can be constructed as the following:

$$H_{\text{IMU}} = \begin{bmatrix} H^q, H^\omega, H^{\alpha,p,v}, H^a, H^b \end{bmatrix}, \tag{3.39}$$

where

$$H^q = \begin{bmatrix} 0_{3\times4} \\ \frac{\partial(R_{\text{IMU}}R^T(\boldsymbol{q})(\boldsymbol{a}+\boldsymbol{g}))}{\partial\boldsymbol{q}} \end{bmatrix}, \tag{3.40}$$

$$H^\omega = \begin{bmatrix} R_{\text{IMU}} \\ 0_{3\times3} \end{bmatrix}, \tag{3.41}$$

$$H^{\alpha,p,v} = \begin{bmatrix} 0_{6\times9} \end{bmatrix}, \tag{3.42}$$

$$H^q = \begin{bmatrix} 0_{3\times3} \\ R_{\text{IMU}}R^T(\boldsymbol{q}) \end{bmatrix}, \tag{3.43}$$

and

$$H^b = \begin{bmatrix} I_6 \end{bmatrix}. \tag{3.44}$$

Each time an IMU measurement is obtained, the EKF performs a time update and a measurement update. A pose estimate is obtained in the form of a state matrix given in Equation 3.33. Furthermore, each time a Kalman estimation cycle is performed, the measurement values, the state vector $\boldsymbol{x}$, and the error noise covariance matrix P are inserted in a first in, first out (FIFO) buffer. This buffer has a fixed length. When this length is reached, values at the front will be discarded while new values are added. The exact length of the FIFO will depend on the camera estimates' pre-measured latency. This buffer will be used to reset the state of the EKF and extrapolate the state once a delayed camera measurement is obtained and used.

Incorporating camera pose estimates

The frequency of incoming camera measurements depends on the frame rate of the used camera. A typical camera frame rate is 30 frames per second (fps), or one frame approximately every 33 ms, which is significantly less than the IMU measurements. As an initial step after obtaining a camera measurement, every frame needs to be processed with the algorithm described in subsection 3.3.2 to obtain a pose estimate for the object. The camera pose estimates consist of an object position estimate and a quaternion orientation estimate. Both obtaining the camera frame and processing the frame with this algorithm introduce latency in the obtained pose estimates. This latency is presumed to follow a Gaussian distribution. The mean of the Gaussian distribution needs to be taken as the expected delay of each camera pose estimate to be used in the algorithm. Whenever a camera pose estimate is obtained, the algorithm needs to perform several steps as outlined in the flow chart. The first step is to locate the actual time of the camera measurement by subtracting the expected latency from the timestamp of the camera measurement. With the actual timestamp, the location of this measurement among previous IMU measurements can be located in the buffer. Once the location is known, the EKF can be reverted to one state previous to the camera measurement time. This can be done with the state vector and the error covariance matrix in the buffer. Now the EKF can be run with the camera estimate. For the time update, the system dynamics equation is the same as was given in Equation 3.34. Therefore, the Jacobian will be calculated as in Equation 3.35. The difference lies in the measurement update. The measurement model of the camera pose estimate is linear and can be specified as:

$$h_{\text{cam}}(\boldsymbol{x}_k) = \begin{bmatrix}\boldsymbol{q}_k & \boldsymbol{p}_k & \boldsymbol{v}_k\end{bmatrix}^T . \tag{3.45}$$

$\boldsymbol{q}_k$ and $\boldsymbol{p}_k$ are direct outputs of the algorithm described in subsection 3.3.2. Nevertheless, the instantaneous linear velocity measurement of the camera, $\boldsymbol{v}_{cam}$, needs to be calculated separately. This is obtained as the following:

$$\boldsymbol{v}_{cam} = \boldsymbol{v}_k - [(\boldsymbol{p}_k - \boldsymbol{p}_{cam})/\Delta T_{cam}] \tag{3.46}$$

Note that in Equation 3.46, $\boldsymbol{v}_k$ represents the linear velocity vector of the state vector at the actual camera measurement time. $\boldsymbol{p}_k$ is the position measurement at the actual camera measurement time, and ΔT_{cam} is the time that has passed from the previous actual camera measurement time until the current. With Equation 3.46, the bias that the velocity vector has at the actual camera measurement time is corrected and used as a camera velocity measurement.

Like with the IMU measurements, the Jacobian of the camera measurement model needs to be calculated. Due to Equation 3.45 being a linear equation, the Jacobian H_{CAM} can be calculated as the following:

$$H_{\text{CAM}} = \begin{bmatrix} I_4 & 0_{4\times6} & 0_{4\times6} & 0_{4\times9} \\ 0_{6\times4} & 0_{6\times6} & I_6 & 0_{6\times9} \end{bmatrix} . \tag{3.47}$$

Now that the equations are complete, the camera measurement update can be performed in the correct state of the system. This will effectively correct the state estimates from the past when the actual camera measurement was taken. With this correction, the filter now needs to extrapolate the state estimates to the current time to get a corrected estimate of the current state of the system. This is done by applying the buffered IMU measurements to the corrected EKF state sequentially until the current time. With this step, a corrected pose estimate is obtained, the system can continue to receive new measurements, and the object is tracked.

Chapter 4

Methodology

As highlighted in chapter 1, object tracking is a fundamental aspect of any MMT system. Ideally, such a system would be versatile enough to function in entirely unknown remote environments. It would be able to explore these environments and monitor any changes that occur with objects within them. To address this complex problem, we have to begin with a basic approach. We have posed the critical question regarding the scenario where we have complete control over our working environment: What advantages can we leverage to enhance the feasibility, reliability, and performance of object tracking? This question is the starting point of this thesis and aims to lay the foundations for developing even more effective and dependable MMT object-tracking systems.

Working in an environment where we can manipulate and fully understand every aspect simplifies several challenges. For instance, having the ability to use specific objects or embed tracking devices inside them gives us a significant advantage. Additionally, we can begin our efforts assuming that an object's movement is limited to a specific number of degrees of freedom. This approach allows us to tackle the task in stages, gradually progressing until we achieve tracking across all six degrees of freedom (6 DoF). This step-by-step method helps streamline the process, making the goal of comprehensive object tracking more attainable.

In addition to developing a high-performance object tracking solution, another important objective of this project has been to ensure cost-effectiveness and ease of implementation. Throughout the electronics design phase of the project, we have carefully considered these aspects, which have influenced the choice of algorithms we could employ. By achieving these objectives, we aim to deliver a reliable, cost-effective, and high-performance tracking solution. This solution is designed to integrate seamlessly with our group's ongoing research and to support other MMT projects requiring an object-tracking solution.

The development of our project has gone through several phases, each expanding the system's capabilities. Initially, we aimed to create an object pose estimation system that embedded a microprocessor with an IMU inside the tracked object. However, we encountered challenges due to sensor errors and drift. To tackle these issues, we have expanded the system by incorporating correction inputs. These inputs, derived from an independent object tracking solution such as a camera or depth sensor, can compensate for drift. At the same time, the IMU can improve the latency and noise associated with other tracking modalities. Ultimately, we have advanced our design to a comprehensive system capable of achieving complete, low-latency object tracking. This system combines a smartphone camera with an IMU embedded within the tracked object, offering a robust solution for accurate tracking.

A single computer has been used for all computational tasks, network interfacing tasks, and simulations that were required during this project. The hardware specifications of this computer can be found in Appendix A. Whenever this computer is used in the text, it is referred to as 'the computer'. Furthermore, the object that is being tracked by the object tracking system and has the electronics embedded inside it is referred to as 'the object'.

The following section lays out the project's initial requirements and general design decisions. Next, several simulation and visualization tools developed and employed throughout the process are explained. The latter sections in this chapter detail the progression of the project's electronics design, followed by the development of the object tracking algorithms.

4.1 Requirements

To better understand and define our project, we set specific requirements and limitations from the start. Our initial plan was to track a single object within a predetermined, empty space. This allowed us to start with a manageable scenario that could be expanded upon later if necessary. Specifically, we decided to track a cube object with 12 cm sides. We chose a cube for its predictable behavior when manipulated, such as pushing, ensuring it would not roll, and that it could maintain a stable orientation on a set path. Furthermore, a cube can easily be restricted to a path along one of its principal axes. This way, we can experiment with tracking a single degree of freedom. As a next step, it is possible to push the cube on a flat surface without toppling. This allows for tracking in 3 degrees of freedom, adding orientation to 2d positional tracking. Finally, it can also be utilized for full 6 DoF tracking. The dimensions of the tracking area were partly determined by the haptic device used by our research group, a Novint Falcon [45]. The Novint Falcon is a haptic device with a controller that can move in 3D space and can provide accurate force feedback. The device can be used by the operator as a controller and in the remote domain as a robot actuator. The device occupies a space of 22.86 cm^3. The controller arm can extend between 3.5 cm and 15.5 cm above its base, which proves ideal for manipulating a cube of the determined dimensions.

An important project requirement is that the object should be tracked with ultra-low latency. Latency in this context refers to the time that passes between the sensing of the object and the processing of a pose estimation. Research on the round trip latency required for a bilateral teleoperation system was found to be in the order of 1 ms [1]. Having this in mind, we set out to reach this goal as closely as possible. This requirement directly dictates the frequency that the object tracking algorithm should be, which is at least 1 kHz with processing latency under 1 ms.

During the project, we also set flexible criteria regarding the project's cost. We determined that the final product should consist of easily accessible off-the-shelf components that are also cost-efficient.

4.2 Tools for Testing and Simulation

During the system's development, we simultaneously focused on the electronics design and the development of the tracking algorithms. This periodically required the simulation of hardware or sensor outputs to test the algorithms, observe performance problems, and quickly prototype solutions. To aid in this process, we have developed several tools. The first tool allows the visualization of the outputs of the tracking algorithms by simulating the tracked object and rendering its movements in a 3D model. Following this, we have created virtual versions of the sensors used throughout the project. These virtual sensors can generate data either on a preprogrammed path or through interaction with a simulation that runs on a game engine. Together, these tools form a complete test bed that has been important in refining the development process and allowing iterative testing of our tracking system.

4.2.1 Visualization Tool

The visualization component of the testing tools has been developed using a 3D game engine, specifically the Unity game engine [46]. Utilizing a 3D game engine is an effective

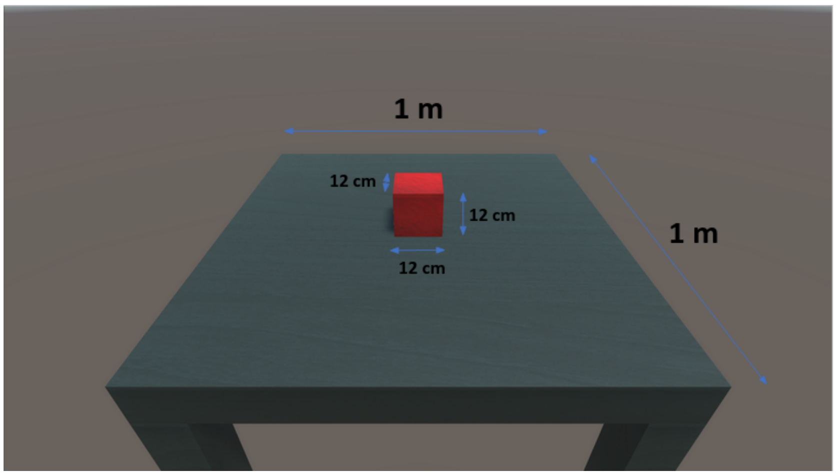


Figure 4.1: **The Unity scene created for the visualization tool. The red cube is the object that is being tracked, representing the real-life tracked cube object. The object is placed on a flat surface representing the real-life tracking environment. The object can be moved around in the simulation based on obtained tracking estimates to visualize and test tracking algorithms or it can be moved around in the simulation to create movement data.**

way of visualizing and simulating the physical environments relevant to our project. They provide a way to easily create 3D models for objects and environments and are equipped with a physics engine that can be adjusted to meet our needs. The Unity game engine has been chosen due to its user-friendly interface, which is intuitive but has powerful incorporated tools. It is documented well since it is widely used, which makes picking it up easier. Furthermore, it incorporates a sufficiently accurate physics engine that played a crucial role in developing the virtual sensor, which will be explained in the next section. Our objective for the visualization tool was to digitally replicate the tracking environment and objects accurately, allowing these digital replicas to closely mimic their real-world counterparts. For this purpose, we created a Unity scene featuring a flat tabletop with 1 m sides and placed a cube in the center as the starting point. The created scene can be observed in Figure 4.1.

To add functionality within the game engine, scripts are utilized. Several scripts are written and attached to the objects to manipulate the objects within the created virtual environment. Initially, our goal was to visualize the outputs of the tracking algorithms by having the objects in the scene move according to these outputs. We achieved this by having a C# script that reads tracking estimates from a CSV file and replicates the movements recorded in the file, which can be obtained from experiments with the actual tracked object. Next, we aimed to visualize the tracking algorithm outputs in real time. Initially, we established a serial connection to communicate the outputs from the object to the computer. As we progressed in the project, we moved to wireless communication by sending data packets over a WiFi connection based on a UDP socket. We implemented a script that could read from a specific terminal to which the device was connected and designed a communication protocol for this purpose. In the wireless implementation, the script was used to read a specific port instead.

With this implementation, the object's pose in the virtual model represented the pose calculated with the algorithm used at the time. This visualization not only aided in identifying trends and problems encountered during the development of the tracking algorithms but also became a vital component of the virtual test tools explained in the next section. We significantly expanded functionality by integrating the visualization system into our test tools. It offered a more comprehensive and effective tool for developing and refining our tracking solutions.

4.2.2 Virtual Sensor

After visualization had been established, the following functionality that was implemented was the virtual sensor. The concept behind virtual sensor is to allow for testing tracking algorithms without the need to fully implement them on the microprocessor initially. This approach provides the advantage of easily tweaking sensor parameters, which facilitates experimentation with the effects of these parameters on tracking performance. Moreover, even after algorithms are implemented on the microprocessor, they often require rigorous calibration and configuration to function correctly. In such scenarios, the ability to quickly test the algorithm without the time-consuming setup process is highly beneficial. Lastly, debugging and data generation for specific object movements is significantly simplified.

Therefore, the goal was to create a program capable of taking perfect object poses from a simulation and delivering sensor data with adjustable parameters as output. Initially, the sensor to be virtualized was the IMU. This required knowledge of the object's acceleration and angular velocity in its body frame. We have taken advantage of the tracking environment modeled for the visualization tool to achieve this. The position and orientation are monitored when a simulation is run in the Unity game engine. Since the object pose is known, angular velocity can be calculated using the orientation angles, and the acceleration can be determined by double derivating the displacement and adjusting for body rotation using the orientation to obtain acceleration in the body frame. Once the perfect acceleration and gyroscope values are obtained that simulate an IMU at the object's center, noise and other parameters can be added to produce realistic sensor data. In the later stages of the project, camera data had to be simulated as well. For this, instead of simulating an actual camera, the pose of the virtual objects was taken, and sensor noise was added to simulate the output of a camera tracking algorithm.

The C++ programming language has been chosen to build the program. The communication between Unity and the program is established via an internal UDP link on a socket, and a communication protocol is established that can send the necessary data between programs.

4.3 Object Hardware Design

4.3.1 Object Electronics

We started our project by identifying the hardware components required to build the system. For the electronics, we would need an Inertial Measurement Unit (IMU) to measure forces, a wireless transmitter device such as a Bluetooth transceiver or a WiFi module for communication between the object and the computer, a microcontroller capable of driving the IMU and the wireless device as well as processing the algorithms, and a battery pack to power the entire device. Additionally, it was necessary to design the casing for the cube object, which is discussed later in this section.

The first critical component to be selected was the IMU due to the low-latency requirement. We found that common commercially available IMUs, with sampling rates up to a couple of kHz, offered a suitable low-latency solution. We settled on a 1 kHz sampling rate as a sufficient starting point, given the 1 ms sampling period requirement. A typical IMU consists of an accelerometer and a gyroscope. Still, we opted for an additional magnetometer, which did not significantly increase the price of the sensor and could be beneficial for orientation estimation. After comparing various IMUs, we chose the TDK InvenSense MPU9250, which was suitable for our application and readily found in our office. The MPU9250 is a 9-axis IMU that includes an accelerometer with an output data rate (ODR) of up to 4 kHz, a gyroscope with an ODR of up to 8 kHz, and a magnetometer. It is important to note that the MPU9250 has been marked as end-of-life by its manufacturer and is not recommended for new designs. However, we did not consider this a significant issue due to its wide usage among hobbyists and ample market supply.

Next, we needed to select a suitable microcontroller to read the IMU data and run the tracking algorithms. The requirements for the microcontroller included an SPI communication interface for sufficient bandwidth for reading data from the IMU, sufficient processing power to handle the pose estimation algorithms, ample documentation and library support for ease of development, and a small form factor to fit within the cube's dimensions. We decided to use the Raspberry Pi Pico microcontroller board [47], which we already had experience with. The Pico features an RP2040 microcontroller chip with a dual-core Arm Cortex-M0+ processor that can go up to 133 MHz. It comes with 264 kB of SRAM and 2 MB of onboard flash memory, in case any tracking algorithm has large memory requirements. Additionally, the Pico provides 26 multi-function GPIO pins and several communication interfaces, including 2 SPI busses. Furthermore, the board is extensively documented and has robust library support. The compact size of the Pico measures 51 x 21 mm, ensuring a good fit inside the cube object. The board also has an onboard power management unit that operates within 1.8 V and 5.5 V DC, allowing it to be easily powered by AA batteries. Although we had not yet determined the pose estimation algorithms, the Pico's specifications indicated it could drive the IMU and wireless device while having spare peripherals if needed. Some initial research has also shown that the M0+ architecture can run similar orientation estimation algorithms [48], which invoked confidence in our decision.

Before designing other system components, such as wireless communication, we tested the compatibility of the IMU and microcontroller combination. Our first objective was to be able to read the IMU sensors with a data rate of 1 kHz. After connecting the two components on a breadboard and powering the device via USB to the computer, we started experimenting. After setting up the SDK, we confirmed that the sensors could be read at the desired rate. We have also learned how to configure the IMU for different sensitivities and data rates. Although the other sensors worked as expected, we encountered an issue with obtaining reliable readings from the magnetometer within the MPU9250, which we found houses two different dies in a single package. After significant effort, we read the magnetometer at a 100 Hz frequency through a special pass-through mode using internal I2C communication. I2C has a significantly slower clock; therefore, the magnetometer could be read at only 100 Hz, which did not pose a significant issue for our purposes as it is not the primary sensor to be used.

With the sensors functioning correctly, we could begin experimenting with algorithms. We started by implementing a complementary filter. The goal was to confirm we could use the sensor data in an estimation algorithm with the required data rate. The success of this initial algorithm implementation at 1 kHz frequency allowed us to proceed with selecting the remaining electronic components. We focused on wireless communication to transmit the object tracking estimates to the computer. One straightforward solution we considered was to purchase the RP Pico W, a variant of the RP Pico we were using. The Pico W is equipped with an onboard single-band 2.4 GHz wireless interface. This option would have allowed us to seamlessly transfer our existing knowledge of the microcontroller and IMU interfacing while incorporating wireless capabilities. Unfortunately, supply chain issues resulted in delivery times of up to several months, leading us to explore alternatives. During our search, we discovered the nRF24L01+ 2.4 GHz Transceiver module. This module appeared to meet the project's requirements according to its documented specifications. This module has an on-air transmission bandwidth of 2 Mbps. To ensure compatibility, we conducted a preliminary calculation considering the module's primary purpose, which is to transmit object pose estimates from the object to the computer. Each pose estimate includes three coordinates and three orientation values, represented by six floating point numbers (4 bytes each), in addition to a timestamp represented by a 4-byte unsigned integer. This results in a minimum payload of 28 Bytes.

The NRF24 can send packets of 32 Bytes, fitting our payload comfortably. We aimed

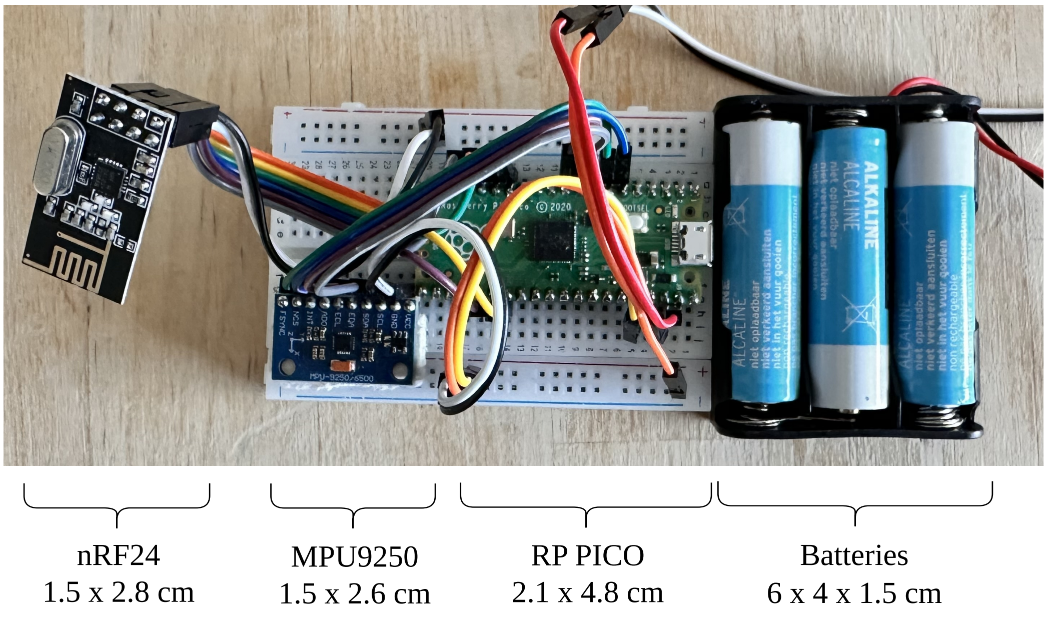


Figure 4.2: **The discrete electronics for the object implemented on a breadboard with component names and dimensions. The nRF24 is the wireless module that transmits tracking estimates. The MPU9250 is the IMU. The RP Pico is the microcontroller. The battery pack consists of 3 AA batteries.**

for a transmission frequency of 1 kHz, which, with our payload, amounts to 28kBps-well within the module's bandwidth. Despite packet handling overhead, which is managed by the device, practical data rates will be lower. However, further research confirmed that data transfer rates up to 46.5 kBps are achievable, equating to 1452 payloads per second. These rates meet our requirements and even allow for a higher transmission frequency.

Additionally, the nRF24 module is very cost-effective and capable of operating both as a transmitter and receiver. For our system, we configured the object as a transmitter and added another Pico with an NRF24 as the receiver connected to the computer. After acquiring and setting up the system, we confirmed that the calculated data rates were achievable with the correct configuration, finalizing the system design. A block diagram illustrating the system design can be seen in Figure 4.3. The transmitter side is embedded in the object, while the receiver side is the microcontroller connected to the computer.

After determining the necessary components and finalizing the design, the system was assembled on a breadboard, which allowed for testing the compatibility of the fully integrated system. The test setup, along with its dimensions, can be seen in Figure 4.2.

However, before moving forward with evaluating the completed hardware design, we decided to pivot our hardware strategy from the discrete design we had developed to an integrated, single-board approach. The primary motivation for shifting to a single-board approach was the realization that our hardware would be embedded within an object that is often in motion. This makes mechanical stability a critical concern. A single-board solution significantly reduces the risk of mechanical issues compared to a discrete design where the components are typically connected with jumper cables or through hand-soldering. Both methods introduce potential points of failure, particularly in dynamic environments. Another consideration was the form factor. Despite careful selection to ensure that the separate components could fit within the intended cube object, assembling them on a prototype board or PCB would result in a much larger footprint than a pre-assembled single-board configuration. This larger size would make integration into the cube more challenging. Lastly, ease of use and replicability were critical factors in our decision. One of the project's objectives was to create a system using off-the-shelf components that

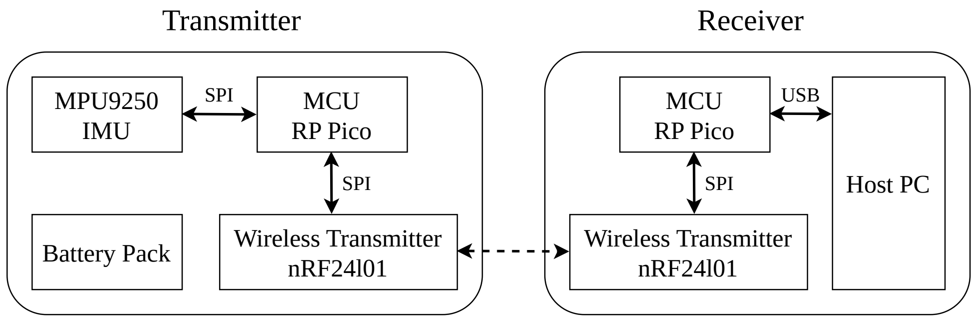


Figure 4.3: **Block diagram of the tracking hardware setup with the RP Pico. The transmitter side consists of the individual hardware components that produce tracking estimates. The estimates are wirelessly transmitted to the receiver side, where a host PC receives and processes the obtained tracking estimates.**

could be easily set up and replicated. A commercially available single board is better aligned with this goal. Hence, we searched for a microcontroller board that ideally met all the previously set component requirements. Finding a board with the exact combination of features was unlikely. Therefore, certain compromises were made during the board's selection process.

Ultimately, we arrived at the Arduino Nano RP2040 Connect [49] for its use of the same chip as the previously utilized Pico, which we anticipated would simplify development due to our familiarity with the hardware and its capabilities. The Nano has a 6-axis LSM6DSOXTR IMU on board, featuring an accelerometer and gyroscope with ODR of up to 6.6 kHz, significantly surpassing our 1 kHz requirement. This IMU also has built-in hardware filters, decreasing the need for additional software filtering of the sensor data. Wireless connectivity is provided by a Nina W102 uBlox module, supporting both WiFi and Bluetooth. This enables a direct computer connection, simplifying the data transfer process. Arduino also has a library available to drive the chip called WIFININA, which makes interfacing with the chip trivial. The Arduino allows an input voltage range from 5 V to 21 V, which offers flexibility in terms of power sources. A compromise in selecting the Arduino is the absence of a magnetometer, which, despite its potential to increase yaw estimation accuracy, presents calibration challenges and sensitivity to magnetic disturbances, especially in indoor settings where the system is planned to be used. A render of the Arduino Nano RP2040 Connect (inside the designed case) can be seen in Figure 4.4.

Limitations of the Arduino Upon finalizing the hardware configuration, we faced certain performance limitations, including slower data reading speeds from the IMU and reduced processing performance compared to the previous hardware setup. The slower IMU reading speeds were attributed to the RP2040's connection to the IMU via I2C instead of SPI. We have circumvented this problem by overclocking the I2C bus. While the bus is rated for a maximum of 400 kbit/s, the setting has been set to 1 Mbit/s to achieve the required speed we have also attained with the previous approach. Although this approach is generally not recommended due to possible stability concerns, we have not encountered any issues on multiple devices during the project and thus approved this approach. Furthermore, we identified the cause of reduced processing performance as the inclusion of the MBED-OS RTOS, which is pre-installed on the Nano and runs the Arduino core. It differs from the bare-metal programming approach of our previous design. We were unable to uninstall MBED-OS without losing access to essential drivers and, thus, opted for a third-party RTOS [50] that offered a closer performance to bare-metal by reducing overhead and enabling the use of the processor's second core. This mitigated the performance issues to a certain degree. Nevertheless, adjustments in the system design had

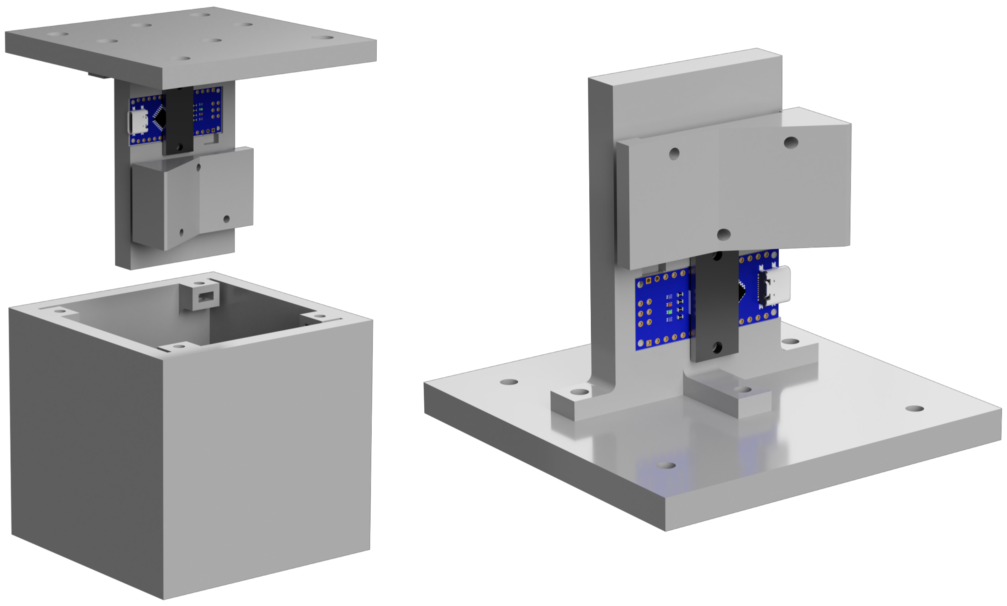


Figure 4.4: **A render of the designed cube object and the lid design. On the left, the lid can be seen lifted above the bottom side of the case. On the right, the components on the lid are visible. The lid has an attached sword that holds the Arduino Nano and a battery pack. The lid can be screwed to the bottom of the case to form a cube object with embedded electronics.**

to be made. These adjustments to address hardware limitations are detailed in section 4.4.

4.3.2 Object Casing

During the process of developing a case for the object, we evaluated several options before arriving at a decision. When we started looking into options for our cube's casing, we initially considered purchasing an off-the-shelf case and modifying it to fit our needs. However, it soon became apparent that finding a suitable pre-made case was not feasible, leading us to discard this option. Instead, we decided to design and 3D print our own case. This decision does not stray from our goal of using easily accessible, off-the-shelf components for the project since 3D printers have become a staple of almost any research facility, and the design files are sufficient to print the case and make modifications if needed.

The dimensions of the cube object have already been established. Therefore, the task is to find a way to embed the electronics inside the object. Given our decision to use a cube with 12 cm sides as the first object, we needed a design that would integrate the electronics, ensuring they fit well inside the object and could be mounted rigidly to avoid disturbances in sensor readings such as vibrations. Additionally, the case needed to be lightweight and easily manipulated by the Novint Falcon. To address these requirements, we developed a cube with a removable lid design, where the electronics are mounted on a shield that attaches to the lid. The design can be seen in Figure 4.4. This configuration facilitates easy access to the electronics from the case when necessary. Moreover, the lid and the shield that holds the electronics are designed as separate components, allowing the shield to be attached to various objects that can be designed later for different experimental setups. The next step involved selecting an appropriate locking mechanism to secure the lid to the cube. Initially, we considered an annular locking mechanism for its ease of use to put on and off but ultimately rejected it due to potential fragility and manufacturing challenges. Self-locking clips were also dismissed for similar reasons. We decided on screws as the locking mechanism due to their simplicity, ease of design and use, rigidity, and wide

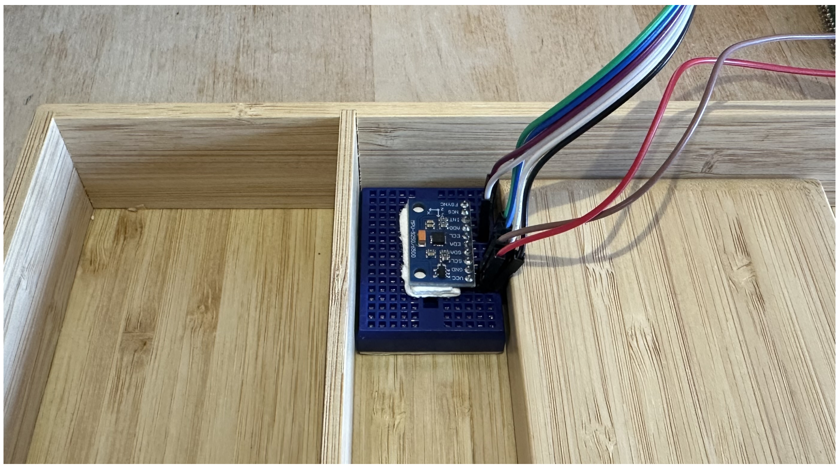


Figure 4.5: **Experimental setup to test the drift performance of the sensor algorithm. The MPU9250 IMU is rigidly attached to a small breadboard (blue). The breadboard is fixed into place by a contraption. The experiment starts in the position shown in the picture. The breadboard and IMU are moved randomly and reinserted into the contraption to execute the drift test.**

availability.

Another critical consideration was ensuring the object's center of mass was close to its geometric center to prevent unpredictable movement during interaction with the object. This was achieved by carefully weighing and positioning the electronics and the battery housing and adjusting the infill rate for the 3D printing process to counterbalance any asymmetry caused by the design. Following the design phase, we successfully printed the parts for the cube. Additionally, we designed another cylindrical object that can utilize the same shield as the electronics, although this design was not printed during the project.

4.4 Tracking Algorithms

4.4.1 IMU Tracking

We began our project with the objective of tracking the pose of an object by utilizing data from an IMU embedded within the object. The pose of an object is defined by its position and orientation, which collectively represent six degrees of freedom (DoF). Our approach to tackling this problem was to achieve incremental tracking from 1 DoF to 6 DoF. In the early stages, after acquiring the electronic components as mentioned in section 4.3 and confirming the Pico's capability to read data from the MPU9250 at a frequency of 1 kHz, we tested whether the processor had enough processing power to run estimation algorithms at the desired frequency. Our initial literature review on object tracking algorithms presented several promising options. As explained in chapter 2, a majority of similar applications use complementary and Kalman filters, and some use particle filters. Particle filters are effective for systems with non-linear dynamics but require significant computational and memory resources due to the need for numerous particles to represent the system's state. Given our goal of running the application on a modest processor, we decided against using particle filters. The initial algorithm we tried is a simple complementary filter. This algorithm calculates the pitch and roll angles of the device by integrating a weighted sum of the gyroscope and the angles derived from the gravity vector components on each axis. The successful implementation of this algorithm allowed us to estimate the device's roll and pitch angles, confirming the processor's capacity to manage this light computational load. Nevertheless, the implemented algorithm was limited in that it could not accurately

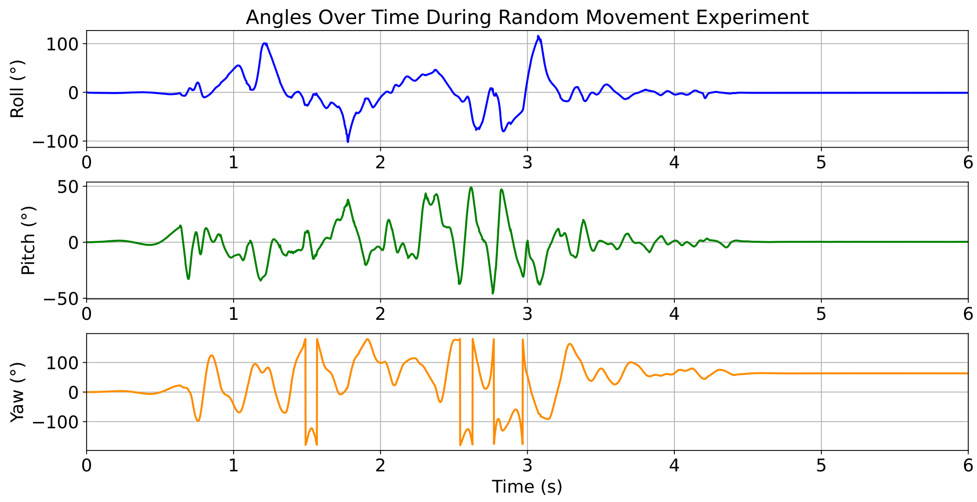


Figure 4.6: **Graphs showing the result of the random movement experiments done with the complementary filter to test drift performance. The graphs show the Roll (blue), Pitch (green), and Yaw (orange) angle estimations of the complementary filter run on the microcontroller.**

calculate the yaw angle due to the orthogonality of the gravity vector to the yaw axis, and magnetometer readings were not utilized. However, a simple test demonstrated the algorithm's effectiveness under static conditions. The algorithm accurately estimated the correct roll and pitch angles when the device was placed on a flat surface. Furthermore, a random movement test has been done with the setup shown in Figure 4.5. The IMU starts in the shown contraption and then is subjected to a random movement. Afterward, the IMU is placed back in the contraption to the exact same starting position, and the angles are tracked during this motion. The results of these random movement tests can be seen in Figure 4.6. We anticipated a drift in the angle estimates due to error accumulation in the integration process, but the accelerometer's gravity vector calculations successfully corrected this drift for the pitch and roll angles by utilizing the gravity vector. Nevertheless, as can be seen in Figure 4.6, while the roll and pitch angles successfully return to a 0° after the random movement test, the yaw angle is subject to drift and remained at an angle offset of 60° after being returned to the contraption.

Moving forward, we explored a more sophisticated algorithm for orientation estimation, the Madgwick filter. This filter estimates orientation by fusing data from the accelerometer, gyroscope, and magnetometer through a gradient descent algorithm, aiming to minimize orientation error. It is a complementary filter like the previous algorithm but offers higher accuracy and includes magnetometer data for yaw angle estimation. Since the Madgwick algorithm is open source, a robust C implementation has been found easily on GitHub [51].

The next step involved testing this implementation on the Pico to assess its feasibility on the system. The algorithm presented a straightforward interface, accepting sensor data and the sampling period as input. To evaluate the implementation, we conducted two similar basic tests as we did with the complementary filter: one with the sensor on a flat surface to observe static drift characteristics and another involving random movements to understand drift after dynamic activity and assess the out-of-the-box performance of the algorithm. During static testing, the angles initially started from the correct values of 0° but exhibited a drift of approximately 5° per second. Similarly, while the angles started correctly in the dynamic case, they did not return their original values after movement, instead settling at significantly different angles. Although better results were expected from the Madgwick filter than were obtained with the simple complimentary filter, the outcomes indicate notable drift under both static and dynamic conditions. This suggests the need for further experiments to characterize and mitigate this drift. However, further experimentation has been paused as we concluded that the results could likely be improved through better sensor calibration and filter parameter optimization. Acknowledging these challenges, we shifted our focus to developing a solution for the position-tracking components, initially attempting the basic approach by integrating acceleration data twice for displacement estimation. However, this method quickly proved problematic due to drift resulting from the accumulation of errors. We then explored the zero velocity update (ZUPT) algorithm [52], commonly used in gait analysis. This algorithm assumes there will be stationary periods (zero velocity) in the movements and applies corrections to reduce the drift. We adopted this technique by identifying stationary periods and calculating the norm of the acceleration vector. A few assumptions needed to be made for this logic to work. The accelerometer senses the gravity vector directed outwards from the earth's center at 1 G. If a force not opposite to the gravity vector is applied to the object, the norm of the total acceleration will be larger than 1 G. Hence, the zero velocity potential update can be applied when a norm less than a certain threshold is calculated. Nevertheless, this approach only works for 3 DoF, as when the object is in free fall, the norm will also be under 1 G while the object is still moving. The implementation showed a significant improvement, yet drift remained an issue during dynamic movements. We recognized that optimizing the performance of both the Madgwick and ZUPT algorithms could potentially be achieved with better sensor calibration, additional filtering, and algorithm configuration adjustments. However, it became clear that relying solely on an IMU for object pose tracking would not meet the stringent requirements of our project. Even with improvements, the system's performance would only be tolerable for a limited time before requiring a reset due to excessive drift. In response, we began developing a new high-level system design, incorporating improvements to address the limitations encountered.

4.4.2 IMU Tracking with Correction Inputs

IMU-based object pose tracking operates on the dead reckoning principle, requiring the object to start from a predetermined pose. The system then calculates displacement and rotation from the initial pose to determine the object's new pose. However, this method is susceptible to drift over time. RGB cameras and depth cameras, on the other hand, offer the ability to estimate an object's absolute pose relative to the camera's position, regardless of the object's initial pose. This capability significantly reduces the impact of drift, making these devices ideal for providing correction inputs to our system. Nevertheless, using cameras as external input introduces another set of challenges, including noise in the estimates due to the limited resolution of these devices, the reliance on 2D to 3D transformations, more complex computations, and added delay due to processing. In response to these challenges, the second iteration of our system was designed to leverage the strengths of both IMU data and external correction inputs, aiming to offset their respective weaknesses. This version processes the same raw IMU data while also incorporating modeled inputs from external sources, such as delayed and noisy pose estimates. We did not focus specifically on camera tracking itself. Instead, we attempted to design the system to be generic enough to accommodate any correction input.

We started this second phase by concentrating on how to integrate frequently updated pose estimates from IMU data, which drifts over time, with pose estimates that are less frequently updated, drift-free, noisy, and delayed due to processing. We found several studies proposing the use of a multi-rate Kalman filter to fuse data with varying update rates and delays, such as [53] and [54]. Based on these insights, we set out to develop a system design that combines pose estimates from individual sensors using a multi-rate Kalman filter with a general system design shown in Figure 4.7. In this system, a single multi-rate Kalman filter fuses the high-frequency orientation outputs of the Madgwick filter with the position outputs of the Kalman filter and, on top of that, corrects itself with a lower frequency correction input. Furthermore, the prediction step of the position

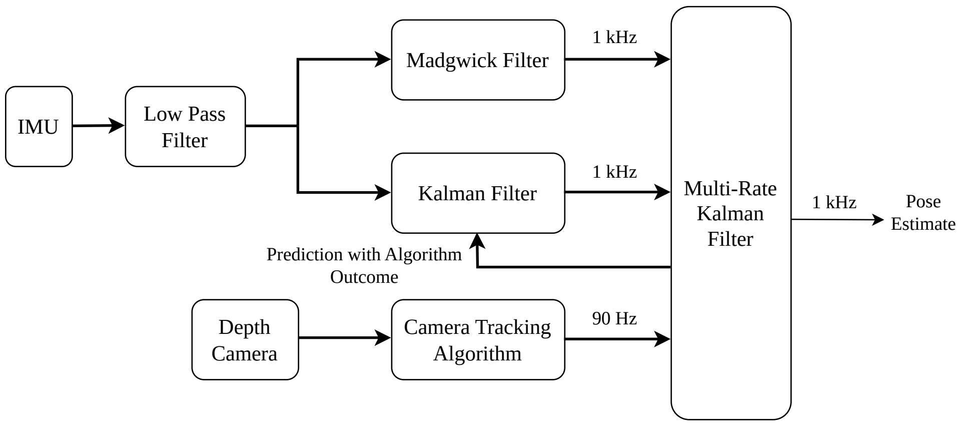
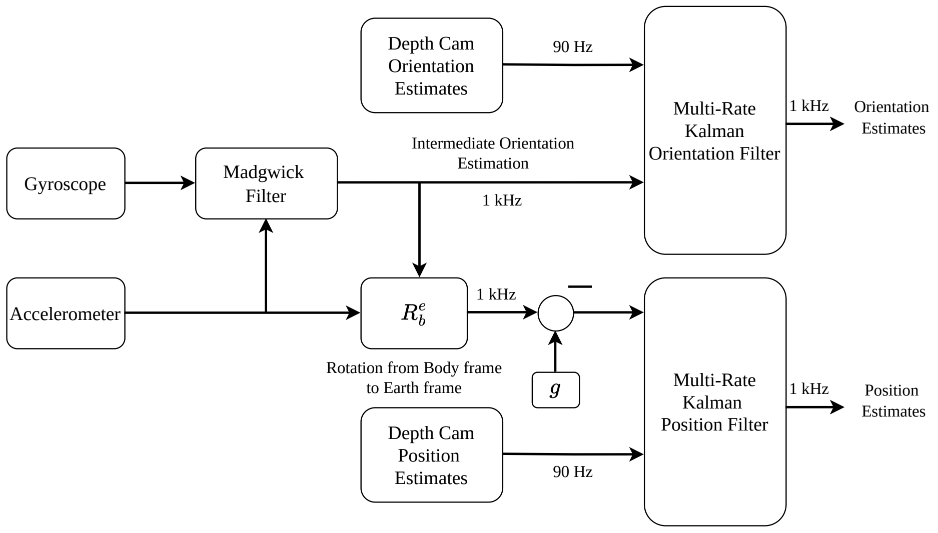


Figure 4.7: **Initial high-level system design of the second phase. A multi-rate Kalman filter is used to fuse the high-frequency orientation estimates of the Madgwick filter, the high-frequency position estimates of the Kalman filter, and the low-frequency camera pose estimates to obtain high-frequency pose estimates of the tracked object.**

Kalman is performed with the position estimates obtained from the multi-rate Kalman filter. The gravity vector corrections for accelerometer measurements are handled in the filter. Initial experimentation of the approach using the virtual test setup showed a major issue with the correction of the gravity vector, leading to significantly diverging position estimates. This led to a secondary design where the multi-rate Kalman filter is split into two distinct filters to estimate the orientation with one and the position of the object with the other. The high-level block diagram of this design can be seen in Figure 4.8. The advantage of this design is that the gravity vector subtraction is handled explicitly before the accelerometer measurements are used in the filter before the multi-rate Kalman filter and, thus, is not affected by inaccuracies introduced by the initial position Kalman filter stage. This approach showed improved performance over the initial design in tests done in the virtual test setup.

However, as explained in subsection 4.3.1, a transition to an integrated electronics board introduced performance challenges, requiring a shift in our approach. Initially, we implemented the Madgwick algorithm on the microprocessor, but due to these new limitations, we could only achieve half of the processing rate of the previous setup, about 500 Hz. Considering the remaining filters that still needed to be implemented, we decided to offload all computational tasks, including those involving IMU data, to a computer. With this change, the Arduino embedded inside the object was left with the task of sampling the IMU and sending the data to the computer. Following this adjustment, we started implementing the algorithm proposed by [54]. This algorithm proposes a method to fuse frequent state estimates with infrequent and delayed estimates. Our focus was initially on fusing single-degree of freedom (DoF) movement of the position to test the performance of the data fusion with the intention to expand to 6 DoF. Since we did not have a camera setup, all tests for this algorithm were done using the virtual sensors setup explained in subsection 4.2.2. The algorithm proposed in [54] can be used to fuse state estimates, but it does not describe a way to calculate them from raw sensor data. Therefore, we have shifted from the previously utilized ZUPT algorithm to a regular Kalman filter for position tracking with the IMU data as explained in chapter 3. The multi-rate Kalman filter operates in 3 distinct modes based on data availability from the different modalities. Initially, it acts as a regular Kalman Filter when only the estimates from IMU data are available. If a measurement is taken by the second modality but has not yet been calculated, the algorithm continues as a regular Kalman filter but begins iterating on a correction matrix



Depth Cam Orientation Estimates
90 Hz
Multi-Rate Kalman Orientation Filter
1 kHz
Orientation Estimates
Gyroscope
Madgwick Filter
Intermediate Orientation Estimation
1 kHz
Accelerometer
R_b^e
1 kHz
−
g
Rotation from Body frame to Earth frame
Multi-Rate Kalman Position Filter
1 kHz
Position Estimates
Depth Cam Position Estimates
90 Hz

Figure 4.8: **Secondary high-level system design of the second phase. Two separate multi-rate Kalman filters are used for orientation estimation and position estimation. The output of the orientation estimates is directly used to subtract gravity components of the acceleration measurements. Low-frequency correction inputs are fused with high-frequency IMU measurements to obtain high-frequency pose estimates.**

to account for the time elapsed since the second modality's measurement was taken. Upon calculation of the second modality's estimate, this correction matrix is then used to fuse the estimate, taking into account its delay, thereby correcting any bias in the IMU data and extrapolating the second modality's estimate to the relevant time point.

After implementing and testing this algorithm with the virtual setup for a single axis of position, we observed that the tracking seemed accurate visually. However, the tracking estimates undesirably incorporated the noise component from the camera estimates, resulting in an interpolated signal that resembled a staircase pattern. The exact issue was difficult to pinpoint but may have stemmed from incorrect implementation or misconfiguration of the state dynamics within the multi-rate Kalman filter. Given the challenges encountered with this complex approach, we opted for a more straightforward method to fuse two distinct types of signals. The first signal comprised estimates calculated from IMU data, sampled at approximately 1 kHz. These estimates tend to drift over time but are accurate in the short term. The second signal consisted of estimates obtained from camera data, which, although less frequent at approximately 90 Hz and subject to delay, are less accurate in the short term but do not exhibit long-term drift. This strategy aimed to leverage the strengths of both data sources to achieve more reliable tracking estimates. In this approach, we used a smoothing filter for the camera estimates. By applying an exponentially weighted Moving Average (EWMA) filter [55], we aimed to represent the camera estimates as a delayed but unbiased signal of the ground truth. This method proved more successful than the multi-rate Kalman approach, allowing us to remove the bias from the frequently updated, accurate, yet biased IMU estimates. Despite its success in bias removal, this approach introduced additional delay in the camera measurements, complicating bias corrections during rapid dynamic movements and negatively affecting our goal for a low-latency system. A significant challenge was the delay of camera estimates, which caused overshooting and undershooting in tracking the object during the dynamic periods. We explored several strategies to address these issues, finding the most effective solution to identify the dynamic periods by checking for significant changes in

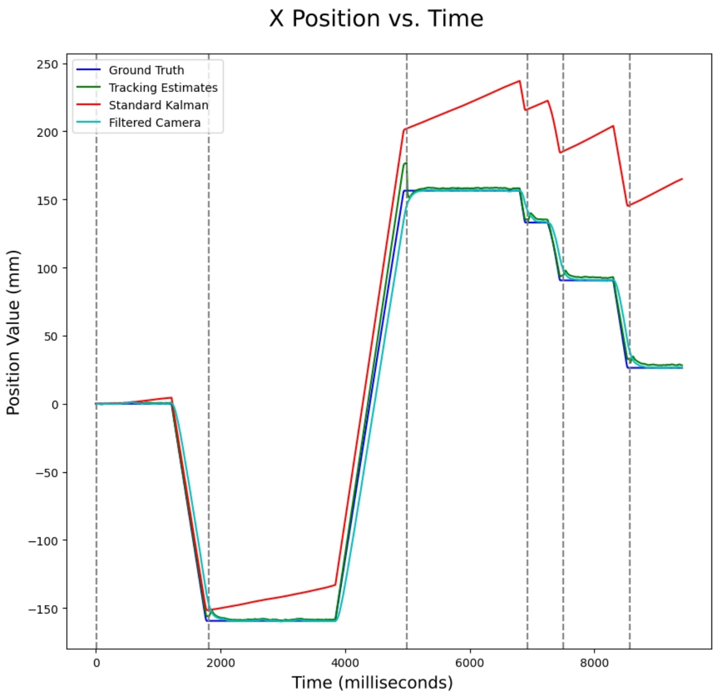


Figure 4.9: **Experimental result from a position tracking experiment using the stationary detection tracking algorithm. The experiment is done in a virtual test setup where the object is moved up and down on the x-axis. The collected data is used to estimate the x position of the object using different tracking algorithms, and the results are compared.**

position estimates from the camera. During these periods, the IMU estimates are not corrected by the calculated camera estimate bias, resuming bias updates once the object becomes stationary again. Figure 4.9 shows the result of a simple experiment where the object is moved up and down along the x-axis in the virtual test environment. The movement is tracked with a standard Kalman filter (red), with filtered camera estimates (cyan), and the fusion of the two with the stationary detection algorithm (green). It can be observed that the IMU estimates closely follow the ground truth but they tend to drift over time. The filtered camera estimates are accurate overall but are delayed with respect to the ground truth. For the fused tracker, the vertical lines in the figure mark the detection of stationary periods, indicating that while the tracker outperforms the individual modalities, accuracy diminishes at the transient points with this method. This approach yielded sub-optimal but reasonable results; however, the method for calculating stationary periods lacked consistent reliability.

In these approaches, we assumed that the correction inputs are generic and could be directly fused with the IMU-derived estimates. This assumption has led to unsatisfactory outcomes, prompting us to reconsider our strategy once again due to practical concerns. Our new approach was to use an actual camera and implement an algorithm that can fuse that data with IMU data in a more complete system that incorporates all components and can capture their interrelations to provide a better and complete way of tracking the object. This approach is laid out in the next section.

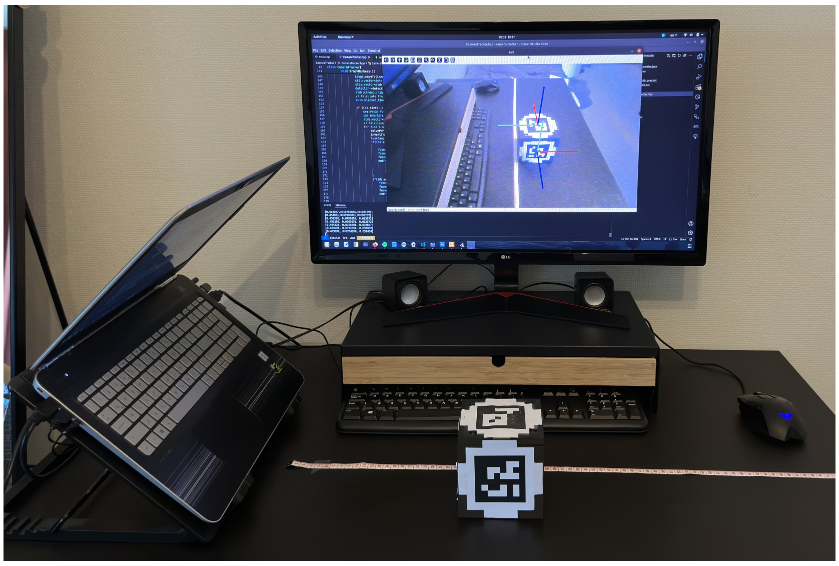


Figure 4.10: **Test setup for camera tracking. The laptop's camera is used to track the position of the cube object with markers on its side. The camera view can be seen on the monitor. During the experiment, the cube is moved on a straight path up and down along its x-axis. The movement of the object is measured using a tape measure attached to the surface.**

4.4.3 Camera and IMU Fused Tracking

We finally decided to transition to a complete system to further enhance our tracking capabilities by integrating a more direct and real-time method of acquiring pose information. Thus, we shifted our development efforts towards creating a complete, self-contained object-tracking system. This comprehensive solution encompasses a dedicated camera setup for visual tracking, an object with the IMU setup inside it, and a tailored tracking algorithm that fuses data from both modalities optimized for our specific use case. By doing so, we retained the system's previous adaptability to fuse various data sources and significantly improved its tracking accuracy and reliability through direct visual inputs.

To achieve this, several steps were required. Initially, we needed to select an appropriate camera, establish a method for obtaining pose estimates with this camera setup, and finalize the algorithm to combine these estimates with IMU data, creating a complete tracking solution. We opted to use a smartphone camera as our camera setup due to the ubiquity of smartphones, their high-resolution cameras, and the fact that they are a cost-effective solution. For transmitting the camera footage to the computer, we utilized an application called IRIUN [56] for its free availability on Android and IOS platforms and its simple use. Since we could receive camera footage on the computer, a way was needed to process this footage. A natural choice was to use OpenCV [44] since it is a freely available, comprehensive computer vision library designed for computational efficiency and real-time applications. With the camera setup in place, we needed to decide what method to use to track the object. We considered two main methods: deep learning-based and marker-based methods. Given the significant computational resources and training data required by deep learning methods and our full access to the object and tracking environment, we chose to use fiducial markers for their efficiency and simplicity. Specifically, we selected AruCo markers due to OpenCV's comprehensive support and the robustness of these markers as indicated by research [57].

We printed a 6 cm by 6 cm AruCo marker and attached it to one side of the object, as seen in Figure 4.10. After setting up the required code for receiving and processing the camera data, we set up a simple test rig to evaluate the position-tracking accuracy of

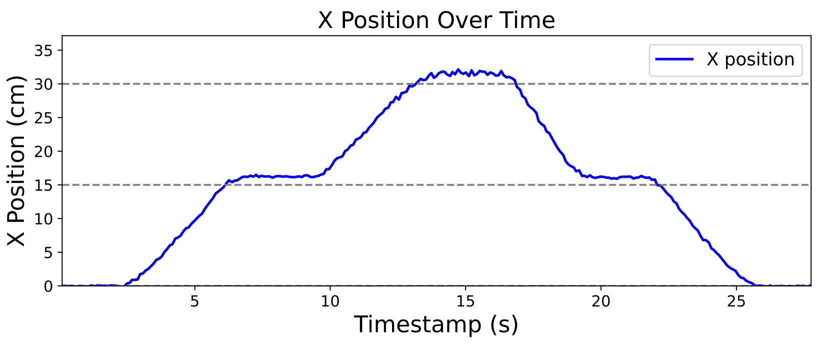


Figure 4.11: **Graph showing test results of a dynamic camera tracking test on a single axis. During this experiment, the object is moved on the x-axis starting from a position at 0 cm. Then the object is moved to 15 cm and 30 cm and back to 0 cm in the same order. The position estimates of the camera tracking algorithm are shown in the graph.**

the system on a single axis. The test setup can be seen in Figure 4.10. Before the tests, the camera was calibrated using the tools also provided by openCV. The tests included evaluating the noise of the position estimates at three different distances from the camera while the object is static, and comparing the accuracy of position tracking by moving the object to three predetermined distances and comparing the estimations to the actual positions. For the static test, the position estimates for the shortest object-camera inter distance of 20 cm, the standard deviation of the estimates has been found to be 0.88 mm, and the furthest test of 60 cm from the camera has shown a standard deviation of 2.11 mm during a 15-second long static test. These low standard deviations for the given distances show that the camera system achieves high precision. Figure 4.11 shows one of the obtained graphs following a dynamic test. In this particular test, the object was moved from a starting x position of 0 mm to 15 cm to 30 cm, as shown by the dotted horizontal lines on the graph, and then moved back in the same order. The vertical axis shows the x position of the marker, while the horizontal axis shows time. As can be seen, the estimates overshoot more the further the object moves from the camera, and the high-frequency noise component increases as well. Nevertheless, when returned to its starting position, the estimates do not drift and end up on the correct value. Several similar tests have resulted in the same findings. This indicates that there is a scaling issue. This issue has been resolved by applying a scaling factor based on the empirical results.

The experiments confirmed that (for 1 DoF) the camera system works and can be utilized further with the remaining system. At this point, the camera tracking was established, and for the IMU estimates, we had the Kalman filter approach. A way to fuse the two modalities reliably still needed to be established. During our search, we found [37], which uses an extended Kalman filter for a very similar setup to ours. In their approach, both the IMU and the camera are embedded in the object (Robot in their case), while in our approach, the camera is external and aimed towards the object with the IMU. Based on this successful approach, we decided to adapt the Extended Kalman Filter to our case. Furthermore, we have used [32] to get more insight into this approach and have finally adapted it to our case, designing a complete Extended Kalman filter for our system that can fuse the two sensors with additional steps that mitigate the latency introduced by camera estimates. This Kalman Filter was detailed in chapter 3.

Chapter 5

Implementation

This section details the implementation of the proposed algorithm in software and its integration with hardware to achieve a functional setup. First, the general system design is explained, followed by a discussion of the system components used in the experimental setup.

5.1 Overall System Implementation

The general system consists of several components. There are two sensor components, the camera and the IMU, and a processing component, the computer. The setup of these units and how they communicate with each other are specified in the following sections.

5.1.1 Camera Setup

The camera being used in the system is a standard smartphone camera capable of taking 30 fps video with a resolution of 1280 x 720. A free-to-use application called DroidCam [58] transfers the camera feed to the central computer. This frame transmission method has been chosen as an alternative to a dedicated webcam or camera to mitigate additional costs since smartphone cameras nowadays have improved significantly, and they are ubiquitous. The footage sent by the application is received by an accompanying desktop application that is mounted on the Linux drivers as a camera device. From there, the application responsible for reading the footage uses the hardware drivers, as would be done like a standard webcam.

The processing of the camera frames is done with the freely available open Computer Vision Library (OpenCV) [44]. This library is chosen due to its extensive and customizable libraries.

Camera frame reception is done with the help of a dedicated thread on the central computer. This thread receives the camera frame and applies the necessary processing to obtain a pose estimate from the frame. When the pose estimate is ready, the estimate is sent to the main thread for processing.

5.1.2 Arduino Firmware

We have chosen the Arduino Nano RP2040 Connect for the setup as explained in chapter 4. The main components of this Arduino used for this project are the CPU, IMU, and WiFi module. The main task of the Arduino is to send the sampled IMU data to the central computer for further processing. Some preliminary filtering and averaging are applied in the Arduino before the data is sent. The IMU is capable of a sampling rate of 6.6 kHz for both the accelerometer and the gyroscope. The tracking system is required to achieve tracking estimates with a 1 kHz frequency. To achieve this, the IMU still samples

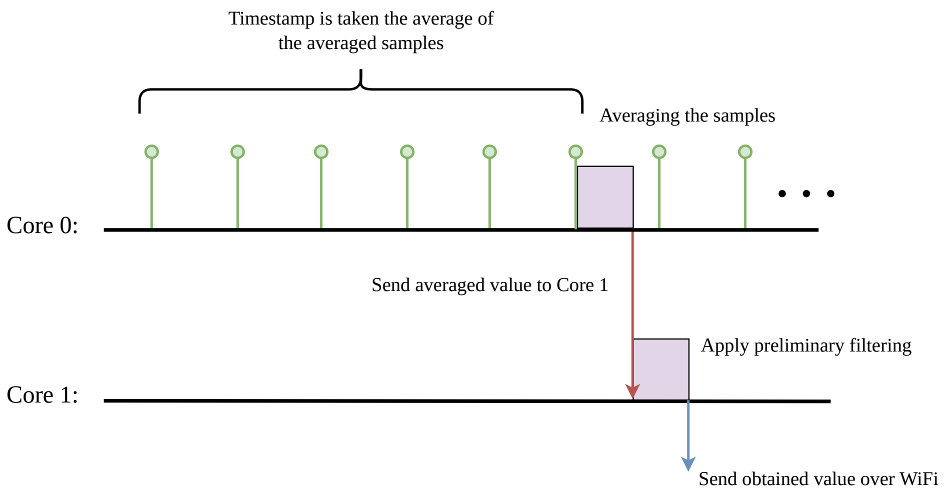

Timestamp is taken the average of
the averaged samples
Averaging the samples
Core 0:
Send averaged value to Core 1
Apply preliminary filtering
Core 1:
Send obtained value over WiFi

Figure 5.1: **Timing diagram showing the operation of firmware IMU sampling. Core 0 samples the IMU at a high frequency and relays the average of the last 6 samples to core 1. Subsequently, core 1 applies preliminary filtering of the samples and sends the output over WiFi to the central computer.**

the sensors with its maximum sampling frequency of 6.6 kHz but does not immediately send all the data. A couple of limitations are involved in sending all data directly if the sampling rate is 6.6 kHz. First, the WiFi module on board has processing limitations and bandwidth issues. Although the WiFi controller is a separate processing unit on the Arduino, the CPU of the Arduino still requires significant overhead to send each packet to the WiFi module, and the WiFi module has limited bandwidth. Since 1 KHz is enough to satisfy our requirements, the process explained in Figure 5.1 has been implemented. The Arduino has two available processing cores. Core 0 is used for sampling the IMU and averaging six samples before sending it to core 1. Averaging is done as a simple noise reduction step to eliminate high-frequency noise in the system. Core 1 is responsible for receiving the data and relaying it to the onboard WiFi module for transmission. The communication between the WiFi module and the central computer is done via UDP protocol. UDP is a sufficient communication method since low latency is required, and the algorithm can compensate for packet loss. Besides the averaging of samples, the IMU has built-in hardware filters that are used to mitigate high-frequency noise. These hardware filters provide an additional low-latency step for filtering.

5.1.3 Software Libraries

OpenCV (version 4.8.0) was used in the final system for image processing and part of the camera tracking functions. OpenCV is a powerful open-source library for computer vision tasks. The Eigen library (version 3.4.0) [59] was also used for linear algebra calculations, such as matrix operations needed in the Kalman filter. Eigen is known for being efficient and easy to use for mathematical computations.

5.2 Test Setup

To analyze the performance of the proposed solution, the following test setup has been prepared: A unique cube object has been 3D printed in the exact dimensions as mentioned in chapter 5, but mounting holes have been placed on the bottom such that the cube can be attached as an end effector to the robot arm. A depiction of the object attached to the

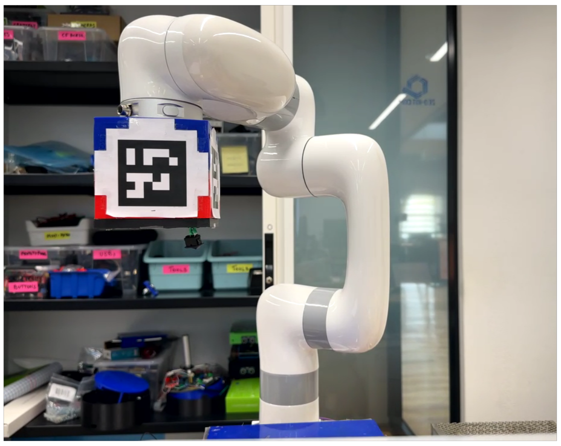


Figure 5.2: **The 3D printed, mountable cube object attached to the end of the robot arm for the experimental setup (left) and the smartphone mounted on a tripod as used in the experimental setup (right). In the experimental setup, the camera is positioned towards the cube. The robot arm moves the cube on preprogrammed paths while the smartphone camera records its movement. The electronics inside the cube simultaneously send IMU data to the central computer. The obtained dataset is used to analyze the performance of the proposed tracking algorithm.**

robot arm can be seen on the left side of Figure 5.2. Furthermore, a smartphone has been mounted on a tripod and directed to the robot arm such that the object is in the frame of its camera. A picture of the mounted phone can be seen on the right in Figure 5.2.

The components of this setup communicate with a central computer via WiFi and ethernet connections. The computer is used to control the robot arm and collect and save the data from the IMU sensor that is sent from the microcontroller. Furthermore, it receives and saves the camera frames that are taken by the smartphone. A connection diagram of the whole experimental setup can be seen in Figure 5.3. The WiFi module on the Arduino inside the cube acts as an access point to which the computer's WiFi module is connected. The computer is also connected to a router via Ethernet. This router also connects to the robot arm via an Ethernet connection, while the smartphone is connected to the WiFi access point created by this router.

The experiments are conducted as follows: The robot arm is controlled using ROS2 [60]. Therefore, the complete data collection program has been written using ROS2 nodes. One node moves the robot on a preprogrammed path. The second node opens a UDP connection with the IMU and records the obtained IMU samples. The third and last node uses openCV to record the video frames taken by the smartphone, which acts as a webcam. A launch file has been written to send synchronization messages to the nodes. Once the system has been set up, a first synchronization message is sent to the IMU and camera nodes to establish connections and record data for three seconds. The recorded data during this phase calibrates the algorithm before the experiments. After three seconds, another message is sent to the robot, IMU, and camera nodes, signaling the start of the actual data collection during the cube's movement. After the movement of the robot arm is completed, the robot node publishes a final message to the data collection nodes to finish the experiments.

Two movements were conducted using the experimental setup. During **Movement 1**, the robot arm moves the cube back and forth 20 cm in the +x direction in the local frame with a sinusoidal speed pattern. During this movement, the cube is held steady in all other axes for both position and orientation. The sinusoidal speed pattern is chosen to better illustrate the tracking performance during an accelerated and decelerated movement. Furthermore, the larger the acceleration, the more effect the delayed camera measurements

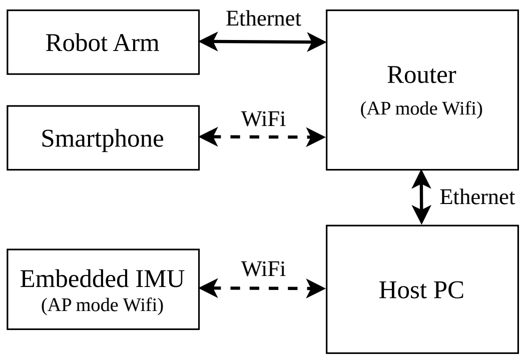


Figure 5.3: **Diagram showing connections between components used in the experimental setup. The robot arm and smartphone are connected to the Host computer via an intermediate router. The embedded IMU inside the cube is directly connected to the host computer by acting as a WiFi access point. Both ethernet and WiFi connections are used in the setup.**

will have on tracking performance. **Movement 2** has been designed to assess the orientation tracking performance of the system. The object position is not changed while a 90° rotation is applied on the yaw angle of the cube object with a sinusoidal speed. All other angles are also held stationary. The sinusoidal speed has been chosen for the same reasons as in the first movement.

During both experiments, the standard settings on the sensors were used. The IMU sends data with 1.14 kHz while the camera takes measurements at 30 fps.

Although the position of the robot's end effector has been logged to obtain a ground truth measurement for the movements during the result analysis, these logs have not been used directly. The reason is that these position measurements are obtained by a forward kinematic function in the robot arm and are not 100% accurate. Instead, to obtain reliable ground truth, the tracking results are compared using the camera footage of the experiments. The captured frames are first filtered using a Sav-Gol filter [61] with a polynomial order of 3 and a window length of 9 to smooth out the noise. Then, the timestamps are adjusted for camera latency to match the IMU data timestamps directly and compared with the forward kinematic logs of the robot arm. The Kalman parameters used during the experiments are the parameters that are specified in Appendix B unless stated otherwise.

Chapter 6

Results

Several experiments have been conducted to analyze the tracking performance of the proposed system. The tracked object has been moved on predetermined paths, and data from the sensors have been collected. The movements are intended to emulate an object in a dynamic environment at the controlled side of a teleoperation system. The proposed algorithm has been run on the collected data to assess tracking accuracy in position and orientation under different conditions, which are explained in the subsequent sections.

The performance of the system is analyzed for the following cases:

1. Comparison of position tracking performance between the proposed system, tracking estimates obtained by camera measurements, and tracking estimates obtained using only IMU estimates.

2. Comparison of orientation tracking performance between the proposed system, tracking estimates obtained by camera measurements, and tracking estimates obtained using only IMU estimates.

3. Comparison of position tracking performance in the presence of increasing camera estimate latency between the proposed system and tracking estimates obtained by camera measurements.

4. Comparison of orientation tracking performance in the presence of increasing camera estimate latency between the proposed system and tracking estimates obtained by camera measurements.

5. Analysis of position and orientation tracking performance with different camera estimate frequencies.

In the 3rd and 4th experiments, artificial camera estimate latency has been added to the datasets obtained in movements 1 and 2 to compare the system's performance under similar conditions. The timestamp of the camera estimates has been adjusted, and the algorithms have been run with these adjusted timestamps. For the 5th experiment, the camera estimate frequencies have been adjusted for each run by decimating the base camera estimates.

6.1 Tracking Performance Analysis

The upper graph in Figure 6.1 shows the position tracking estimate results of different methods for movement 1, while the lower plot shows the absolute error of these estimates. It can be seen that when only IMU data is used to calculate the position tracking estimates, an accurate result of a positional error below 1 cm can be attained for approximately 1.3 seconds. After this time, the position estimates obtained using only IMU data tend to drift drastically due to the accumulation of errors from the double integration of acceleration measurements. The IMU-only estimates are able to follow the direction of the movement until approximately 6 seconds, after which it diverges completely. Another observation of the IMU-only measurements is that a high-frequency error is introduced as time progresses. In the IMU-only Kalman filter, the only measured state concerning position is the acceleration. Hence, the only state that converges to a fixed process noise is that of acceleration, while the velocity and position states continue to diverge as new measurements come in. In time, the filter will trust the acceleration measurements more, resulting in the propagation of acceleration measurement noise to the position state estimates.

In the upper plot of Figure 6.1, it can be seen that the camera estimates follow the ground truth relatively well. The camera estimates do not accumulate drift as is expected. Due to the inherent latency of the camera estimates, it can be seen that the estimates lag behind the ground truth. This lag was observed to be approximately 30 ms. As a result of this lag, an error pattern of the camera estimates can be observed in the lower plot of Figure 6.1. If the object had been moved with a constant velocity, the constant lag would have resulted in a constant error between the camera estimates and the ground truth proportional to movement speed. The speed in this experiment is sinusoidal, which causes the object to reach a speed of 0 at the peaks and valleys of the sinusoid while reaching its largest speed precisely between them. Since the error of the lagging camera estimates is proportional to object speed, a sinusoidal error pattern can be observed in the camera error as well. Another notable feature in the camera position estimates can better be seen in Figure 6.2, which shows a close-up of the first 2 seconds of Figure 6.1. In the upper plot of Figure 6.2, the camera estimates follow a sample and hold signal resembling a staircase pattern. This is because the camera algorithm operates with only 30 Hz. This means that the algorithm has no new information about the object's position after a camera estimate is obtained. Therefore, the estimate remains on the last outcome for approximately 33 ms until a new estimate is received. The effect of lack of estimates can consequently be observed in the error signal of the camera estimates. In the lower plot of Figure 6.2, the error signal of the camera shows a zigzag pattern in combination with its sinusoidal path. The error will naturally increase if the object moves between two camera estimates. Furthermore, this error will be compensated for when a new camera estimate arrives, resulting in a zigzag pattern. In Figure 6.2, it is also visible that when the object is moving slower at second 2, the zigzag pattern diminishes while it is the most prominent at second 1.

In the upper plot of Figure 6.1, the position estimates of the EKF can be seen to follow the ground truth accurately throughout the whole movement. The lower plot shows that the maximum error of the EKF position estimates is a little above 0.2 cm while the average remains around 0.1 cm. Considering that in this experiment, the EKF is using the same camera estimates and imu data as the other algorithms shown in the same figure, it performs significantly better than both methods individually. No drift is observed throughout the movement. Even though the EKF uses the camera estimates with latency, it does not show the same error pattern and effectively uses the camera estimates to correct the drift from the imu data while eliminating the latency they introduce. Although the EKF performs better than both modalities, it is still affected by the accuracy of the individual measurements. This can be seen by comparing the error signal of the EKF and the camera at approximately 2.2 seconds in Figure 6.1. At this point, the camera estimates can be seen to be less accurate than average, and a jump is caused in its error signal. Similarly, at that time, the EKF also suffers from a spike in its error. Looking at the error signal of the EKF in the close-up in Figure 6.2, it can be seen that the EKF error is affected by the frequency of the camera estimates. In between camera estimates, the EKF is prone to drift due to its operation solely on imu data at these intervals, and that is corrected whenever a camera estimate is received and used.

The upper plot in Figure 6.3 shows the resulting orientation estimates of different object pose tracking methods for movement 2. The lower plot shows the corresponding absolute

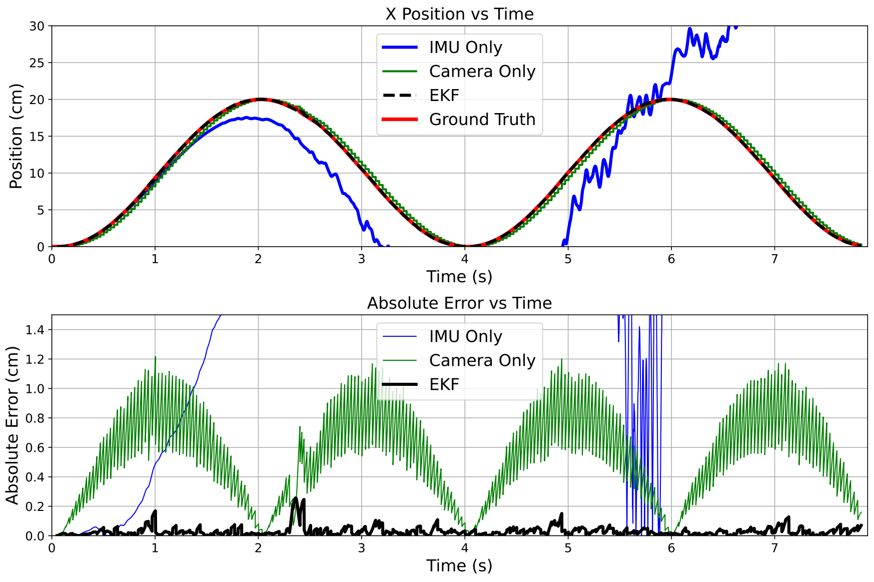


Figure 6.1: **Tracking estimates and absolute error of the object's x position from movement 1. The upper plot shows the estimates over time, while the lower plot displays the absolute error of the position estimates relative to the ground truth. The graphs compare the estimates obtained from three different tracking methods: IMU only, camera tracking, and the proposed EKF. Note the time axis is aligned.**

angle errors of these orientation estimates. Similarly to the position estimates obtained by the IMU-only algorithm, the angle estimates obtained with the IMU-only algorithm tend to drift. These angle estimates are able to accurately trace the ground truth data for approximately 2 seconds. During this interval, the error of these estimates stays around 1°, after which the estimate drift increases.

In contrast to the IMU-only position estimates, the imu-only angle estimates do not diverge completely during the movement and still follow the direction of movement until the end of the experiment. This decrease in drift is caused by the single integration of gyroscope measurements to obtain angle estimates, in contrast to the double integration of acceleration measurements for the position. This also affects the high-frequency error of the IMU-only angle estimates. Furthermore, the less noisy nature of gyroscope measurements also affects this. When looking at the camera angle estimates and its corresponding error in Figure 6.3, similar trends can be seen as were observed in position camera estimates. Due to the sinusoidal rotation speed and the camera estimate latency that has been calculated as 100 ms for this movement, the error of the camera angle estimates also follow a sinusoidal pattern. Due to the camera frequency of 30 Hz, the camera angle estimates also follow a staircase pattern. Consequently, the angle errors also follow a zigzag pattern where, at its height, an oscillation of 1 degree is observed. The angle orientation estimates obtained with the EKF can be seen to follow the ground truth accurately, similar to the position estimates. The maximum angle error of the EKF estimates is approximately 3.5° during movement 2. This maximum value is observed at second 7 in the lower plot of Figure 6.3. A sudden peak in the camera rotation estimates can be seen. This error in the camera estimate translates to a peak in the EKF estimates. Throughout the movement, the EKF error stays consistently below the camera and IMU errors with a mean value of approximately 0.5°. Overall, it can be concluded that the proposed algorithm can accurately track both the position and the angle of the object by using delayed camera estimates and noisy IMU data. The latency of the camera estimates and the noise in IMU data is effectively eliminated, and the data is fused to obtain accurate tracking.

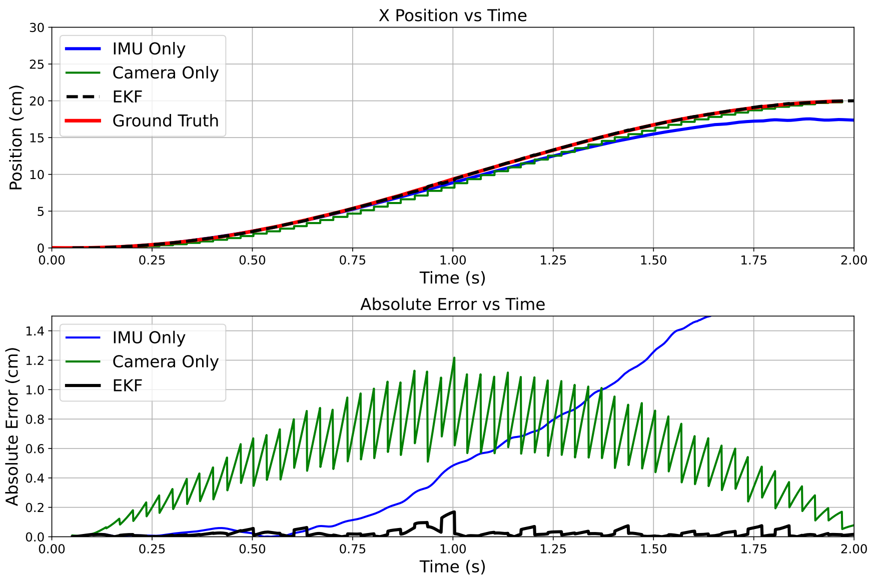


Figure 6.2: **A close-up of the first two seconds of the graphs shown in Figure 6.1. This figure similarly depicts tracking estimates and absolute error of the object's x position from Experiment 1. The upper plot shows the estimates over time, while the lower plot displays the absolute error of the position estimates relative to the ground truth. Note the time axis is aligned.**

6.2 Impact of Camera Latency on Tracking Performance

Figure 6.4 and Figure 6.5 show the experimental results of the latency experiments. In these experiments, the latency of the camera estimates has been artificially increased and used in the tracking algorithms to see the impact camera latency has on tracking performance. In each step, the latency has been increased by 10 ms. While Figure 6.4 shows position error results obtained from movement 1, Figure 6.5 shows the angle error results obtained from movement 2.

In Figure 6.4, the red line with triangle data points shows the absolute median error of the camera estimate tracking method as the latency of the camera estimates increases. When the camera estimate latency is 0, the median error of the camera estimate is also approximately 0 with minimal spread. This indicates that the camera-only method is very successful in accurately and precisely estimating the object's position. As the latency of these estimates increases, the camera-only estimates are still accurate, but when they arrive, the object has moved on. Therefore, the estimation indicates a position in the past. This effectively increases the error for all estimates, especially when a fast movement occurs. Furthermore, an increase in latency will result in an increase in error. This fact can be observed in Figure 6.4. The median error of the camera-only estimates in each step increases linearly, and the error distribution widens as latency increases. Every 10 ms of added camera latency increases the median position error by approximately 1 mm. It should be kept in mind that this linear relation is proportional to the movement velocity of the object, and the mentioned relation is unique to the sinusoidal movement pattern used in movement 1. The increase in error spread as latency increases is also a direct result of the sinusoidal pattern.

The blue line with crossed data points and the green line with the dotted data points in Figure 6.4 show the median error for the position estimates under increasing camera latency for the proposed method. While the blue line indicates the results where the algorithm has been run with the default Kalman parameters, the green line indicates the

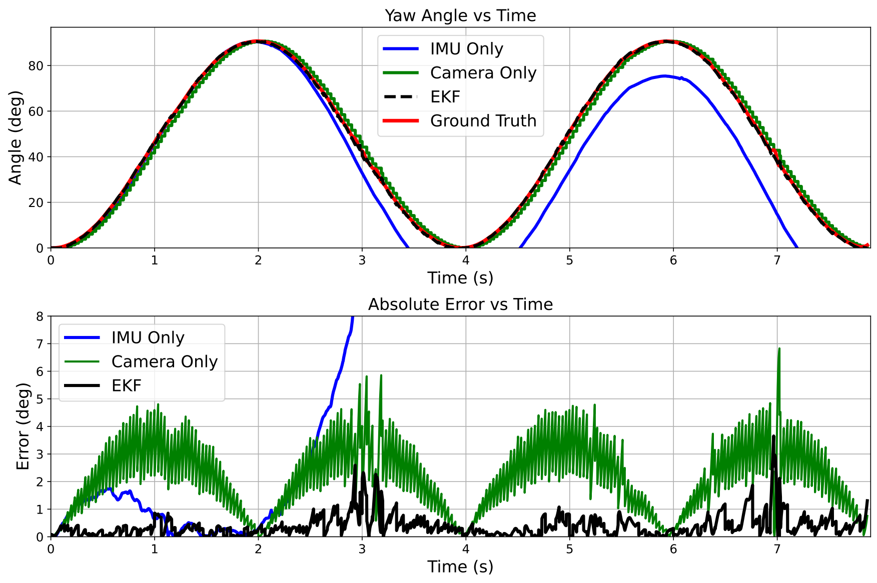


Figure 6.3: **Tracking estimates and absolute error of the object's yaw angle from Experiment 2. The upper plot shows the estimates over time, while the lower plot displays the absolute error of the yaw angle estimates relative to the ground truth. The graphs compare the estimates obtained from three different tracking methods: IMU only, camera tracking, and the proposed EKF. Note the time axis is aligned.**

results where some algorithm parameters have been modified, namely the process noise covariance entries for velocity (Q_v). Q_v equals 0.15 in the default parameters, and for the other run, they have been decreased to 0.0001. It can be seen that the EKF with $Q_v = 0.15$ is able to keep positional tracking error very low in between camera latency values of 0 ms to 160 ms. This window shows A slight increase from 0.2 mm to 1.2 mm. This slight increase can be attributed to the fact that the larger the latency of the camera estimates, the longer the algorithm has to rely on imu data for extrapolation, resulting in a larger drift. After 160 ms of latency an increase in median error to approximately 2.1 mm is observed until latency becomes 190 ms for the EKF with $Q_v = 0.15$. This bump in error signals that the EKF cannot keep up with the camera latency, and drift starts to occur. After 190 ms of camera latency, the error of the EKF with $Q_v = 0.15$ can be seen to be completely diverging. The reason for this is that the filter cannot handle the amount of extrapolation under its current configuration. One reason might be the velocity measurement calculation method from the camera estimates. The amount of extrapolation results in a significant drift between camera measurements, resulting in positive feedback during velocity calculation, causing the filter to diverge.

The green line in Figure 6.4 shows how this effect can be delayed by compromising initial position tracking accuracy. The green error line indicates the results where Q_v has been lowered to a value of 0.0001. This decrease in the velocity process covariance causes the filter to trust the velocity measurements less and rely more on the dynamic model. As can be seen in Figure 6.4, this results in a shift in the divergence point of the filter from 190 ms to 360 ms. Nevertheless, this increased tolerance to camera latency increases overall error for position tracking, especially in the lower latencies. When the blue and green lines are compared between 0 ms latency and 160 ms latency, the filter with default parameters has a constant lower error of approximately 0.2 mm. Figure 6.4 shows that the proposed method can very accurately estimate object position even when using camera estimates that are delayed up until 350 ms. Nevertheless, camera-only measurements are still more accurate if no latency is present than the proposed filter. Judging from the

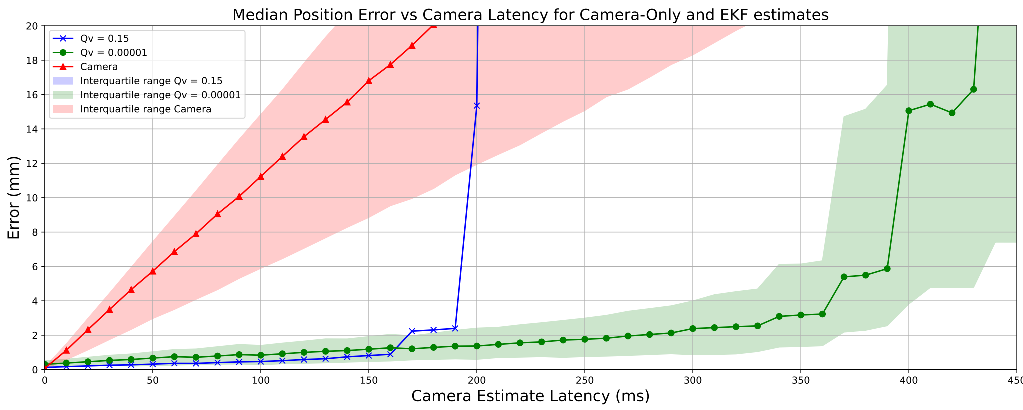


Figure 6.4: **Median absolute position error as a function of camera estimate latency with interquartile ranges, comparing Extended Kalman Filter (EKF) estimates with different settings and camera-only estimates. Qv represents the process noise covariance matrix entries of velocity.**

apparent tradeoff of tolerating latency and decreased accuracy when the results for Q_v = 0.15 and Q_v = 0.0001 are compared, it can be said that it is crucial to optimize and calibrate the Kalman parameters of the proposed filter depending on the conditions of the used measurements. This will result in more accurate filter tracking. Furthermore, if it is impossible to gain camera estimates with a latency of less than 350 ms, camera-only estimates might be preferred over the proposed method.

Figure 6.5 shows the impact of camera estimate latency on orientation tracking for the camera-only method and the proposed method. The camera-only algorithm and the proposed EKF are run on the dataset obtained with movement 2. Similar to position tracking, the angle error of the camera-only method starts with a very low median error and reaches an angle estimation error of 1° when the latency is 20 ms. The median error increases linearly with latency and quickly reaches 5° when the latency is 50 ms. On the other hand, the proposed EKF can keep angle estimation error low until the filter diverges at around 270 ms. Even when camera estimates are delayed by 250 ms, the median absolute angle error of the EKF is 2°, while at the same latency, if only camera estimates are used, the median angle error reaches 14°. Notably, the spread of error distribution for the proposed EKF is narrow until the divergence point. This points out consistent, accurate estimations throughout each run. After the divergence point of 270 ms, the EKF method is no longer usable for angle estimations. After this point, it is preferable to use the camera-only method. This divergence is thought to be caused by similar reasons the position tracking diverges, namely the velocity measurements. Furthermore, at low latencies, it can be seen that the camera-only method outperforms the EKF, and the EKF performance starts overtaking at 10 ms latency. When there is no latency present, although the same camera estimates are used in both methods, the interpolation with imu data by the EKF causes a slight performance drop. At 0 ms latency, the camera median error is measured as 0.07° while the EKF median angle error at that point is measured as 0.4°. All in all, in a range where camera estimates are delayed between 10 ms and 270 ms, the proposed EKF provides reliable and accurate angle tracking and effectively eliminates latency introduced by the camera measurements.

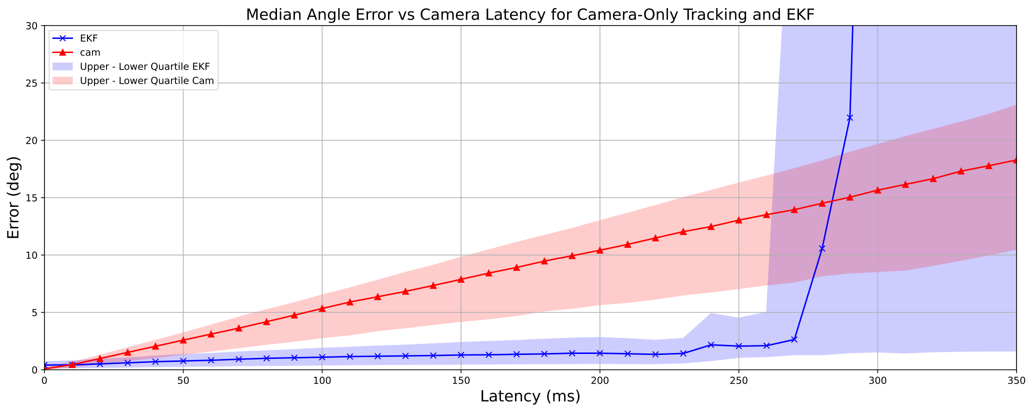


Figure 6.5: **Median absolute orientation error as a function of camera estimate latency with interquartile ranges, comparing Extended Kalman Filter (EKF) and camera-only estimates.**

6.3 Impact of Camera Frequency on Tracking Performance

Figure 6.6 and Figure 6.7 present the results of the frequency experiments. Figure 6.6 shows the median error in position estimates from the proposed EKF applied to the dataset obtained from movement 1. Figure 6.7 illustrates the median error in orientation estimates from the proposed EKF applied to the dataset obtained from movement 2. A range of frequencies is obtained by decimating camera estimates from the original datasets. Since the original camera used in the experiments runs at 30 fps, the graphs show a frequency range from 0.12 Hz to 30 Hz.

When the position error versus camera estimate frequency in Figure 6.6 is observed, it can be seen that the median error follows an exponential decrease with increasing frequency as expected. At low camera estimate frequencies, 0.12 Hz to around 1 Hz, the median position error can be seen to have diverged to above 50 mm. As the camera estimate frequency increases from 0.12 Hz to around 0.8 Hz, there is a notable decrease in the median position error that swings around 5 mm. From 0.8 Hz to 1.1 Hz, this oscillation continues in a decreasing fashion. Beyond 1.1 Hz, the error continues to decrease, and the graph shows a significant drop in error, stabilizing at lower values. The interquartile range is wide at lower frequencies, indicating high variability in the error. As the frequency increases, the variability decreases. This means that at higher frequencies the EKF is able to consistently estimate more accurately. The graph demonstrates that the frequency of camera updates profoundly affects the position-tracking accuracy of the EKF. Using higher camera estimate frequencies of 1.1 Hz and above is beneficial for optimal performance. Nevertheless, beyond 10 Hz, the error does not decrease significantly as it approaches the baseline performance of the tracker.

It should be noted that at frequencies below 1 Hz, the EKF gets camera updates at intervals longer than 1 second. This means that the EKF operates on only IMU estimates for over a second. As is mentioned in section 6.1 The EKF is able to track the position accurately for about 1.3 seconds, after which the estimates diverge. This is in line with the findings in this graph, as it can be seen that the tracking accuracy degrades significantly when the camera estimate frequency is lower than 0.8 Hz (1.25 s intervals).

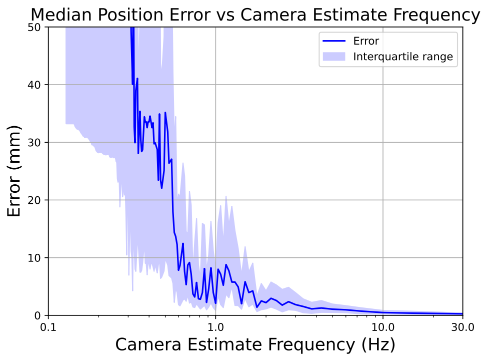


Figure 6.6: **Median absolute position error as a function of camera estimate frequency with interquartile ranges for the proposed EKF. Note the x-axis is in logarithmic scale.**

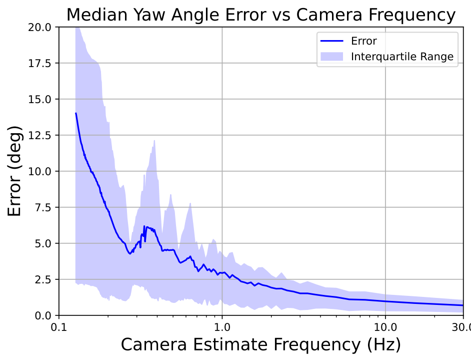


Figure 6.7: **Median absolute orientation error as a function of camera estimate frequency with interquartile ranges for the proposed EKF. Note the x-axis is in logarithmic scale.**

Figure 6.7 shows the median angle error of the EKF for a range between approximately 1.2 Hz to 30 Hz. In general, it can be observed that the median error steadily decreases as the camera estimate frequency increases. At the low-frequency range of around 0.12 Hz, the median yaw angle starts high at approximately 14 degrees. As the camera estimate frequency increases from 0.12 Hz to around 1 Hz, the median error decreases significantly to around 3 degrees. Beyond 1 Hz, the error continues to decrease, stabilizing at lower values. Around 10 Hz and higher, the median yaw angle error levels off to a baseline and stays relatively stable, around 1°. The interquartile range is wide at lower frequencies, indicating high variability in the yaw angle error. As frequency increases, the variability decreases, and the interquartile range narrows significantly. This means that not only does the yaw angle error decrease with higher frequency updates, but the consistency of the error also improves. Concluding from this graph, the camera estimate frequency updates profoundly affect the accuracy of the EKF in tracking the yaw angle of the object. Low-frequency updates result in high and variable yaw angle errors, while high-frequency updates lead to low and consistent errors. This is a similar trend observed for the position estimation errors as discussed. For optimal performance in angle tracking, using higher camera estimate frequencies (10 Hz and above) is beneficial.

It can also be seen that the median error has some high-frequency fluctuations along its path, with a significant dip at around 0.3. The reason for this may be due to the method with which the frequency range is obtained. Although this affects the variability of individual error ranges, it does not affect the general trend observed in the graph.

Chapter 7

Future Work

Several modifications can be made to the proposed system to improve it in certain areas. In the current system, the pose-tracking algorithm runs entirely on an external machine, while the microcontroller inside the object is only used for simple filtering and networking. The subsequent iterations could make better use of the microcontroller resources by incorporating improved preprocessing of the IMU samples before transferring them to increase overall accuracy or improve the networking operations to decrease the latency even further. Similarly, the camera frames are sent directly from a phone with many layers of software stacks (camera application, operating system, etc.) that affect the latency between taking the image and receiving it on the computer. To mitigate this, a custom system could be made with a camera and microcontroller that could be capable of the camera pose tracking algorithms locally for improved latency.

Another improvement could be a complete integration of the system into a working bilateral teleoperation system. Currently, the system has been tested in an isolated environment where the primary component is tracking the object's pose. Bilateral teleoperation systems consist of many complex components that need to work seamlessly together. When all components, including object tracking, are brought together in a comprehensive system, unforeseeable interactions that need to be investigated may occur. For example, other components on the network may limit the bandwidth, or other computational loads may limit performance, which can affect the latency of the system output.

Further research could focus on new system designs. The system developed in this thesis was designed based on specific requirements and assumptions, aiming to create an accurate, low-cost, and accessible solution with low latency. However, the requirement for physical modifications limits the system's applicability to objects that can be altered, reducing its flexibility in real-world scenarios. Therefore, this foundation opens the door to natural next steps. A first step could be eliminating the need for direct access to the tracked object by integrating alternative sensors such as depth cameras or stereo vision systems. Additionally, increasing the number of external sensors can enhance spatial coverage and reduce occlusions. Furthermore, the algorithms can be enhanced to incorporate simultaneous localization and mapping (SLAM), which is crucial for applications where the environment cannot be predefined. Environment discovery not only aids in object tracking but is also essential for achieving full bilateral teleoperation with force feedback. This goal could be reached by employing advanced machine learning techniques, such as convolutional neural networks for object recognition and recurrent neural networks for motion prediction. As discussed in this thesis, these sophisticated algorithms would require more powerful computational resources, such as GPUs, to process the data without compromising latency. Finally, a natural last step would be for the system to fully discover new environments, including tracking non-rigid objects.

Chapter 8

Discussion

The making of this project has not been a straightforward path, characterized by setbacks and changing scopes. As the project progressed, adjustments became necessary, and although these changes presented challenges, they ultimately provided invaluable insights.

In hindsight, a comprehensive literature review before starting the project could have significantly influenced the direction of the work. While it initially seemed time-efficient to explore approaches through hands-on experimentation, a broader and deeper review could have saved significant time in steering the project in better directions and knowing what was possible or not beforehand.

This project underscored the importance of planning in incremental steps. Establishing clear objectives for the overall project and smaller, attainable goals is crucial for effectively achieving the primary outcome. While the project's planning did not always proceed as anticipated, this experience highlighted the value of structured goal-setting at both the macro and micro levels to maintain direction throughout the development process.

The initial approach in the project involved embedding a sensor within an object to track the object under the assumption that this would be a straightforward task. However, this part quickly proved more complex than anticipated. Setting up the electronics and ensuring component integration required substantial time and effort, impacting the project's overall timeline. Further in the project, as more components had to be integrated, other significant integration challenges arose. In future projects, accounting for the complexity of integration early on and avoiding postponing it until the final stages would be essential.

This project made use of multiple sensors and we found out later on in the project their performance is heavily dependent on meticulous calibration. Proper sensor calibration, followed by a validation of both the calibration process and the calibration outcome itself, is essential. Underestimating the importance of this step led to a loss of significant time, as issues arising from calibration inaccuracies were initially attributed to other factors. In future projects, prioritizing thorough calibration and validation procedures would mitigate such delays.

A virtual test setup was developed to streamline algorithm testing, including a simulation environment and a virtual sensor. While the visualization tools of this setup proved very helpful in identifying issues and refining algorithms, the resulting virtual sensor did not closely resemble an actual IMU since the practicalities of an actual sensor are too complex to implement quickly and could be a whole project on its own. Future endeavors would benefit from using existing virtual IMU solutions or focusing only on the visualization component. Similarly, employing the Unity game engine as a physics simulation tool without a good understanding of its limitations resulted in challenges. It would be better to understand the capabilities and limitations of such tools before using them in larger parts of the project.

Throughout the project, numerous approaches and algorithms were explored. While exploring various solutions is essential, switching between methodologies too frequently diverted attention from actual project goals. Several algorithms were ultimately abandoned because they were not feasible or applicable, or initial testing showed undesirable results. A more structured approach would involve thorough research on each algorithm before implementation and working out the implementation to completion before redeeming them unsuitable. This way, fewer and more accurate approaches can be determined.

Achieving minimal latency was a primary objective, yet the challenges presented by managing latency in a complex system were not fully anticipated. Each hardware and software layer contributed minor latencies, which compounded to impact the overall system latency. Additionally, synchronization with time-sensitive measurements became an issue, as even minor divergences led to system failures. This is a point to take into account in later projects.

While these mentioned obstacles stretched our timeline, they also provided hands-on learning experiences that were invaluable in understanding practical system development. Ultimately, we created a robust tracking system that aligns with the project's original goals. Nevertheless, while our system is effective, it has limitations. Due to time constraints, we were unable to test it within a fully operational MMT setup. However, preliminary performance suggests it could be successful in that context. Additionally, we characterized the system based on single-axis evaluations for position and orientation but have not yet assessed it under more complex, multi-axis motions. Further testing with complex movements would provide a more comprehensive understanding of the system's behavior in realistic MMT scenarios. We also did not measure the entire system latency from movement to pose estimation with high accuracy. As a result, a rigorous comparison between our final system and state-of-the-art solutions remains an opportunity for future work.

Chapter 9

Conclusion

Haptic bilateral teleoperations with force feedback is a research field promising to solve many real-life problems. It will be possible to accurately and organically interact with physical environments separated by very long distances. This application is useful in many fields and industries and will impact the daily lives of everyday people. Certain limitations make the realization of this technology challenging today. The main obstacle is data transfer over long distances with minimal delay. In a model-mediated teleoperation setting, haptic feedback is very susceptible to divergences between the remote and local environments, even with small latencies. Therefore, it is crucial that the local model can be updated as fast as possible. The pose of objects in the remote environment needs to be tracked with very low latency and high accuracy.

This thesis explores the question: How can an accurate, low-latency object pose tracking system be set up as accessible as possible with minimal component and time costs? We hypothesized that by leveraging specific assumptions, such as tracking in a known environment where the object can be prepared in advance, we could create a system that performs comparably to state-of-the-art computer vision solutions, which typically require significant computational resources, are costly, and are complex to set up. Our solution involves a specially designed, 3D-printed object with an embedded inertial measurement unit (IMU) and fiducial markers on its surface. A single camera is also used as part of the setup. We developed an algorithm that uses an extended Kalman filter (EKF) as a foundation and effectively fuses IMU and camera data to provide accurate, low-latency pose estimates. This system serves as a drop-in replacement for more advanced tracking setups and enables further research in model-mediated bilateral teleoperation.

A dedicated test setup was created to evaluate the system and the designed algorithm. The purpose of this setup is to generate a dataset containing ground truth data, which is used to assess the accuracy and latency of the object tracking system. In this setup, the object is mounted on a robot arm programmed to perform predefined motions, allowing for the testing of both position and orientation tracking performance. These movements have been recorded with a camera, resulting in a comprehensive dataset for performance analysis.

The designed system is able to keep the absolute position tracking error below 0.2 cm during an 8 second position experiment with a mean error value of 0.1 cm. During the orientation tracking experiment, it is shown that the system is capable of tracking the object orientation with a mean absolute angle tracking error of 0.5°, and the absolute angle error never exceeded 4°.

Using a standard smartphone camera for accessibility introduces significant latency in the camera measurements. Specifically, position and orientation errors were initially evaluated with camera data lagging 30 ms behind the IMU measurements. To address this, the system implements a correction and extrapolation step. Further experiments demonstrated that the algorithm can effectively utilize camera data delayed up to 350 ms. However, a tradeoff between correctable camera latency and estimation accuracy was observed. For camera latencies below 200 ms, the mean position error remained under 1 cm. With an adjustment in algorithm parameters, delays of up to 350 ms were corrected, but the error increased to 2.5 cm.

The system is composed of a 3D-printed object, an off-the-shelf, low-cost microcontroller board with an IMU and networking capabilities, a smartphone camera, and a standard computer to run the algorithm. These accessible components, when integrated, provide a robust and cost-effective tracking solution. Combined with the performance results of the algorithm, which demonstrate accurate pose estimation despite camera latency, we have successfully validated our hypothesis that a low-cost system can perform comparably to more expensive, state-of-the-art tracking solutions.

Overall, we have answered our research question by designing, implementing and testing the object tracking system. We have confirmed our hypothesis by showing that it is possible to create an accessible object pose tracking solution that serves as a practical and efficient replacement for more complex, expensive tracking solutions, enabling further research in real-time applications in haptic bilateral teleoperation with force feedback.

Bibliography

[1] Martin Maier, Mahfuzulhoq Chowdhury, Bhaskar Prasad Rimal and Dung Pham Van. 'The tactile internet: vision, recent progress, and open challenges'. In: *IEEE Communications Magazine* 54.5 (2016), pp. 138–145. DOI: `10.1109/MCOM.2016.7470948`.

[2] Blake Hannaford. 'A design framework for teleoperators with kinesthetic feedback'. In: *IEEE transactions on Robotics and Automation* 5.4 (1989), pp. 426–434.

[3] K Hastrudi-Zaad and Septimiu E Salcudean. 'On the use of local force feedback for transparent teleoperation'. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No. 99CH36288C)*. Vol. 3. IEEE. 1999, pp. 1863–1869. DOI: `https://doi.org/10.1109/ROBOT.1999.770380`.

[4] Jee-Hwan Ryu and Dong-Soo Kwon. 'A novel adaptive bilateral control scheme using dual closed-loop dynamic characteristics of master/slave manipulators'. In: *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*. Vol. 2. IEEE. 2000, pp. 1173–1178. DOI: `http://dx.doi.org/10.1109/IROS.2000.893178`.

[5] Probal Mitra and Günter Niemeyer. 'Model-mediated telemanipulation'. In: *The International Journal of Robotics Research* 27.2 (2008), pp. 253–262. DOI: `https://doi.org/10.1177/0278364907084590`.

[6] Bert Willaert, Jeannette Bohg, Hendrik Van Brussel and Günter Niemeyer. 'Towards multi-DOF model mediated teleoperation: Using vision to augment feedback'. In: *2012 IEEE International Workshop on Haptic Audio Visual Environments and Games (HAVE 2012) Proceedings*. IEEE. 2012, pp. 25–31. DOI: `http://dx.doi.org/10.1109/HAVE.2012.6374429`.

[7] Xiao Xu, Burak Cizmeci and Eckehard Steinbach. 'Point-cloud-based model-mediated teleoperation'. In: *2013 IEEE International Symposium on Haptic Audio Visual Environments and Games (HAVE)*. IEEE. 2013, pp. 69–74. DOI: `https://doi.org/10.1109/HAVE.2013.6679613`.

[8] Xiao Xu, Sili Chen and Eckehard Steinbach. 'Model-mediated teleoperation for movable objects: Dynamics modeling and packet rate reduction'. In: *2015 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE)*. IEEE. 2015, pp. 1–6. DOI: `https://doi.org/10.1109/HAVE.2015.7359474`.

[9] Xiao Xu, Burak Cizmeci, Clemens Schuwerk and Eckehard Steinbach. 'Model-mediated teleoperation: Toward stable and transparent teleoperation systems'. In: *IEEE Access* 4 (2016), pp. 425–449. DOI: `https://doi.org/10.1109/ACCESS.2016.2517926`.

[10] Syeda Nadiah Fatima Nahri, Shengzhi Du and Barend Jacobus Van Wyk. 'A review on haptic bilateral teleoperation systems'. In: *Journal of Intelligent & Robotic Systems* 104.1 (2022), p. 13. DOI: `10.1007/s10846-021-01523-x`.

[11] Eric Marchand, Hideaki Uchiyama and Fabien Spindler. 'Pose estimation for augmented reality: a hands-on survey'. In: *IEEE transactions on visualization and computer graphics* 22.12 (2015), pp. 2633–2651. DOI: `https://dx.doi.org/10.1109/TVCG.2015.2513408`.

[12] Denis Tome, Thiemo Alldieck, Patrick Peluse, Gerard Pons-Moll, Lourdes Agapito, Hernan Badino and Fernando De la Torre. 'Selfpose: 3d egocentric pose estimation from a headset mounted camera'. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.6 (2020), pp. 6794–6806. DOI: `https://doi.org/10.1109/TPAMI.2020.3029700`.

[13] Renshu Gu, Gaoang Wang and Jenq-Neng Hwang. 'Efficient multi-person hierarchical 3d pose estimation for autonomous driving'. In: *2019 IEEE conference on multimedia information processing and retrieval (MIPR)*. IEEE. 2019, pp. 163–168. DOI: `https://doi.org/10.1109/MIPR.2019.00036`.

[14] Claudiu Pozna, Radu-Emil Precup and Péter Földesi. 'A novel pose estimation algorithm for robotic navigation'. In: *Robotics and Autonomous Systems* 63 (2015), pp. 10–21. DOI: `https://doi.org/10.1016/j.robot.2014.09.034`.

[15] Ursula Kälin, Louis Staffa, David Eugen Grimm and Axel Wendt. 'Highly accurate pose estimation as a reference for autonomous vehicles in near-range scenarios'. In: *Remote Sensing* 14.1 (2021), p. 90. DOI: `https://doi.org/10.3390/rs14010090`.

[16] Andrew I Comport, Eric Marchand, Muriel Pressigout and Francois Chaumette. 'Real-time markerless tracking for augmented reality: the virtual visual servoing framework'. In: *IEEE Transactions on visualization and computer graphics* 12.4 (2006), pp. 615–628. DOI: `https://doi.org/10.1109/TVCG.2006.78`.

[17] Jian Guan, Yingming Hao, Qingxiao Wu, Sicong Li and Yingjian Fang. 'A Survey of 6DoF Object Pose Estimation Methods for Different Application Scenarios'. In: *Sensors* 24.4 (2024), p. 1076. DOI: `https://doi.org/10.3390/s24041076`.

[18] Jiale Chen, Lijun Zhang, Yi Liu and Chi Xu. 'Survey on 6D pose estimation of rigid object'. In: *2020 39th Chinese Control Conference (CCC)*. IEEE. 2020, pp. 7440–7445. DOI: `https://doi.org/10.23919/CCC50068.2020.9189304`.

[19] Martin A Fischler and Robert C Bolles. 'Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography'. In: *Communications of the ACM* 24.6 (1981), pp. 381–395. DOI: `https://doi.org/10.1145/358669.358692`.

[20] Pauline C Ng and Steven Henikoff. 'SIFT: Predicting amino acid changes that affect protein function'. In: *Nucleic acids research* 31.13 (2003), pp. 3812–3814. DOI: `https://doi.org/10.1093/nar/gkg509`.

[21] Ping Wang, Guili Xu, Yuehua Cheng and Qida Yu. 'A simple, robust and fast method for the perspective-n-point problem'. In: *Pattern recognition letters* 108 (2018), pp. 31–37. DOI: `https://doi.org/10.1016/j.patrec.2018.02.028`.

[22] Martin Runz, Maud Buffier and Lourdes Agapito. 'Maskfusion: Real-time recognition, tracking and reconstruction of multiple moving objects'. In: *2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2018, pp. 10–20. DOI: `https://doi.org/10.48550/arXiv.1804.09194`.

[23] Chenyi Liu, Fei Chen, Lu Deng, Renjiao Yi, Lintao Zheng, Chenyang Zhu, Jia Wang and Kai Xu. '6DOF pose estimation of a 3D rigid object based on edge-enhanced point pair features'. In: *Computational Visual Media* 10.1 (2024), pp. 61–77. DOI: `https://doi.org/10.48550/arXiv.2209.08266`.

[24] Haowen Sun, Taiyong Wang and Enlin Yu. 'A dynamic keypoint selection network for 6dof pose estimation'. In: *Image and Vision Computing* 118 (2022), p. 104372. DOI: `https://doi.org/10.48550/arXiv.2110.12401`.

[25] Bugra Tekin, Sudipta N Sinha and Pascal Fua. 'Real-time seamless single shot 6d object pose prediction'. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 292–301. DOI: `https://doi.org/10.48550/arXiv.1711.08848`.

[26] Jun Zhou, Kai Chen, Linlin Xu, Qi Dou and Jing Qin. 'Deep fusion transformer network with weighted vector-wise keypoints voting for robust 6d object pose estimation'. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 13967–13977. DOI: https://doi.org/10.48550/arXiv.2308.05438.

[27] Yan Xu, Kwan-Yee Lin, Guofeng Zhang, Xiaogang Wang and Hongsheng Li. 'Rnpose: Recurrent 6-dof object pose refinement with robust correspondence field estimation and pose optimization'. In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 14880–14890. DOI: https://doi.org/10.48550/arXiv.2203.12870.

[28] Manon Kok, Jeroen D Hol and Thomas B Schön. 'Using inertial sensors for position and orientation estimation'. In: *arXiv preprint arXiv:1704.06053* (2017). DOI: https://doi.org/10.1561/2000000094.

[29] Gabriele Bleser and Didier Stricker. 'Advanced tracking through efficient image processing and visual–inertial sensor fusion'. In: *Computers & Graphics* 33.1 (2009), pp. 59–72. DOI: http://dx.doi.org/10.1109/VR.2008.4480765.

[30] Hanna Nyqvist and Fredrik Gustafsson. 'A high-performance tracking system based on camera and IMU'. In: *Proceedings of the 16th International Conference on Information Fusion*. IEEE. 2013, pp. 2065–2072.

[31] Gabriele Ligorio and Angelo Maria Sabatini. 'Extended Kalman filter-based methods for pose estimation using visual, inertial and magnetic sensors: Comparative analysis and performance evaluation'. In: *Sensors* 13.2 (2013), pp. 1919–1941. DOI: https://doi.org/10.3390/s130201919.

[32] Rodrigo Alves Medeiros, Guilherme Araujo Pimentel and Rafael Garibotti. 'An Embedded Quaternion-Based Extended Kalman Filter Pose Estimation for Six Degrees of Freedom Systems'. In: *Journal of Intelligent & Robotic Systems* 102.1 (2021), p. 18. DOI: 10.1007/s10846-021-01377-3.

[33] Kejie Qiu, Tong Qin, Hongwen Xie and Shaojie Shen. 'Estimating metric poses of dynamic objects using monocular visual-inertial fusion'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 62–68. DOI: https://doi.org/10.48550/arXiv.1808.06753.

[34] Armaghan Moemeni and Eric Tatham. 'A framework for camera pose tracking using stochastic data fusion'. In: *2010 2nd International IEEE Consumer Electronics Society's Games Innovations Conference*. IEEE. 2010, pp. 1–7. DOI: https://doi.org/10.1109/ICEGIC.2010.5716876.

[35] Jason R Rambach, Aditya Tewari, Alain Pagani and Didier Stricker. 'Learning to fuse: A deep learning approach to visual-inertial camera pose estimation'. In: *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE. 2016, pp. 71–76. DOI: https://doi.org/10.1109/ISMAR.2016.19.

[36] Rudolph Emil Kalman. 'A new approach to linear filtering and prediction problems'. In: *Journal of Basic Engineering* 82.1 (Mar. 1960), pp. 35–45. DOI: https://doi.org/10.1115/1.3662552.

[37] Arif Tanju Erdem and Ali Özer Ercan. 'Fusing inertial sensor data in an extended Kalman filter for 3D camera tracking'. In: *IEEE Transactions on Image Processing* 24.2 (2014), pp. 538–548. DOI: https://doi.org/10.1109/TIP.2014.2380176.

[38] Sebastian OH Madgwick, Andrew JL Harrison and Ravi Vaidyanathan. 'Estimation of IMU and MARG orientation using a gradient descent algorithm'. In: *2011 IEEE International Conference on Rehabilitation Robotics*. Ieee. 2011, pp. 1–7. DOI: https://doi.org/10.1109/ICORR.2011.5975346.

[39] Sergio Garrido-Jurado, Rafael Muñoz-Salinas, Francisco José Madrid-Cuevas and Manuel Jesús Marín-Jiménez. 'Automatic generation and detection of highly reliable fiducial markers under occlusion'. In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292. DOI: https://doi.org/10.1016/j.patcog.2014.01.005.

[40] Satoshi Suzuki et al. 'Topological structural analysis of digitized binary images by border following'. In: *Computer vision, graphics, and image processing* 30.1 (1985), pp. 32–46. DOI: https://doi.org/10.1016/0734-189X(85)90016-7.

[41] Nobuyuki Otsu. 'A threshold selection method from gray-level histograms'. In: *IEEE transactions on systems, man, and cybernetics* 9.1 (1979), pp. 62–66. DOI: https://doi.org/10.1109/TSMC.1979.4310076.

[42] Kenneth M Dawson-Howe and David Vernon. 'Simple pinhole camera calibration'. In: *International Journal of Imaging Systems and Technology* 5.1 (1994), pp. 1–6. DOI: http://dx.doi.org/10.1002/ima.1850050102.

[43] Ethan Eade. 'Gauss-newton/levenberg-marquardt optimization'. In: *Tech. Rep.* (2013). URL: https://mat.uab.cat/~alseda/MasterOpt/optimization.pdf.

[44] G. Bradski. 'The OpenCV Library'. In: *Dr. Dobb's Journal of Software Tools* (2000).

[45] Novint Technologies Inc. *Novint Falcon Haptic Device*. https://www.novint.com/. 2024.

[46] Unity Technologies. *Unity*. Version 2023.2.3. 2023. URL: https://unity.com/.

[47] Raspberry Pi Foundation. *Raspberry Pi Pico*. https://www.raspberrypi.com/. 2024.

[48] N. Büscher, D. Gis, V. Kühn and C. Haubelt. 'On the Functional and Extra-Functional Properties of IMU Fusion Algorithms for Body-Worn Smart Sensors'. In: *Sensors (Basel, Switzerland)* 21.8 (2021), p. 2747. DOI: 10.3390/s21082747.

[49] Arduino. *Arduino RP2040 Connect*. https://www.arduino.cc/. 2024.

[50] III Earle F. Philhower. *arduino-pico*. https://github.com/earlephilhower/arduino-pico. 2023.

[51] x-io Technologies. *Fusion*. https://github.com/xioTechnologies/Fusion. Accessed: 15-06-2023. 2023.

[52] Rahul P Suresh, Vinay Sridhar, J Pramod and Viswanath Talasila. 'Zero Velocity Potential Update (ZUPT) as a Correction Technique'. In: *2018 3rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)*. 2018, pp. 1–8. DOI: 10.1109/IoT-SIU.2018.8519902.

[53] Alireza Fatehi and Biao Huang. 'Kalman filtering approach to multi-rate information fusion in the presence of irregular sampling rate and variable measurement delay'. In: *Journal of Process Control* 53 (2017), pp. 15–25. ISSN: 0959-1524. DOI: https://doi.org/10.1016/j.jprocont.2017.02.010. URL: https://www.sciencedirect.com/science/article/pii/S0959152417300367.

[54] Chaitanya Jugade, Daniel Hartgers, Sajid Mohamed, Dip Goswami, Andrew Nelson, Gijs van der Veen and Kees Goossens. 'Improved Positioning Precision using a Multi-rate Multi-sensor in Industrial Motion Control Systems'. In: *2023 European Control Conference (ECC)*. 2023, pp. 1–8. DOI: 10.23919/ECC57647.2023.10178138.

[55] S. Crowder and S. V. Wiel. 'Exponentially Weighted Moving Average (EWMA) Control Chart'. In: (2008). DOI: 10.1002/9780470061572.EQR248.

[56] Iriun Oy. *IRIUN*. https://iriun.com/. 2024.

[57] Michail Kalaitzakis, Brennan Cain, Sabrina Carroll, Anand Ambrosi, Camden Whitehead and Nikolaos Vitzilaios. 'Fiducial markers for pose estimation: Overview, applications and experimental comparison of the artag, apriltag, aruco and stag markers'. In: *Journal of Intelligent & Robotic Systems* 101 (2021), pp. 1–26. DOI: 10.1007/s10846-020-01307-9.

[58] DEV47APPS. *DroidCam*. `https://droidcam.app/`. 2023.

[59] Gaël Guennebaud, Benoît Jacob et al. *Eigen v3*. 2010. URL: `http://eigen.tuxfamily.org`.

[60] Steven Macenski, Tully Foote, Brian Gerkey, Chris Lalancette and William Woodall. 'Robot Operating System 2: Design, architecture, and uses in the wild'. In: *Science Robotics* 7.66 (2022), eabm6074. DOI: `10.1126/scirobotics.abm6074`. URL: `https://www.science.org/doi/abs/10.1126/scirobotics.abm6074`.

[61] Abraham Savitzky and Marcel JE Golay. 'Smoothing and differentiation of data by simplified least squares procedures.' In: *Analytical chemistry* 36.8 (1964), pp. 1627–1639.

Appendix A

Hardware and Device Specifications

A.1 Computer Specifications

The technical specifications of the computer platform utilized throughout this thesis and referred to throughout as 'the computer', are as follows:

- CPU: Intel® Core™ i7-7700HQ CPU @ 2.80GHz $\times$ 8
- RAM: 15.5 GiB
- GPU: NVIDIA GeForce GTX 1050 Mobile
- Operating system: Ubuntu 20.04.6 LTS

Appendix B

Parameters

B.1 Kalman Filter Parameters

This appendix specifies the numerical values of the Kalman filter parameters that were used in the proposed algorithm during the experiments.

The initial state vector:

$$\boldsymbol{x}_0 = \begin{bmatrix}\boldsymbol{q}_0 & \boldsymbol{\omega}_0 & \boldsymbol{\alpha}_0 & \boldsymbol{p}_0 & \boldsymbol{v}_0 & \boldsymbol{a}_0 & \boldsymbol{b}_{G0} & \boldsymbol{b}_{A0}\end{bmatrix}^T . \tag{B.1}$$

where,

$$\boldsymbol{q}_0 = \begin{bmatrix}1 & 0 & 0 & 0\end{bmatrix}^T , \tag{B.2}$$

$$\boldsymbol{\omega}_0, \boldsymbol{\alpha}_0, \boldsymbol{p}_0, \boldsymbol{v}_0, \boldsymbol{a}_0 = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T , \tag{B.3}$$

$$\boldsymbol{b}_{G0} = \begin{bmatrix}0.21554229 & -0.2368477 & -0.04746255\end{bmatrix}^T , \tag{B.4}$$

$$\boldsymbol{b}_{A0} = \begin{bmatrix}0.0090883 & 0.00892818 & 0.01125091\end{bmatrix}^T . \tag{B.5}$$

The initial error covariance matrix:

$$P_0 = \begin{bmatrix}0.001 * I_{25\times25}\end{bmatrix} . \tag{B.6}$$

The process noise covariance matrix:

$$Q = \begin{bmatrix}diag\{10^{-4} \times 4, 0.00047 \times 3, 10^{-4} \times 3, 1 \times 3, 0.15 \times 3, 300 \times 3, 10^{-9} \times 6\}\end{bmatrix} . \tag{B.7}$$

The IMU measurement noise matrix:

$$R_{n,\text{IMU}} = \begin{bmatrix}diag\{10^{-3}, 10^{-3}, 10^{-2}, 0.001218429, 0.001218429, 0.001218429\}\end{bmatrix} . \tag{B.8}$$

The camera measurement noise matrix:

$$R_{n,\text{CAM}} = \begin{bmatrix}diag\{10^{-3} \times 4, 10^{-1} \times 6\}\end{bmatrix} . \tag{B.9}$$

The IMU rotation matrix:

$$R_{\text{CAM}} = \begin{bmatrix}1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1\end{bmatrix} . \tag{B.10}$$

B.2 Camera Parameters

The intrinsic camera parameters that have been obtained with the OpenCV ChArUco calibration tool for the camera used in the experiments are as follows:

$$K = \begin{bmatrix} 8.5556992272374\text{e}{+}02 & 0.0 & 6.3461190199486\text{e}{+}02 \\ 0.0 & 8.5532559004904\text{e}{+}02 & 3.6655589985425\text{e}{+}02 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}. \tag{B.11}$$

The distortion coefficients are obtained as the following:

$$D_{coeff} = \begin{bmatrix} 1.98107812833317741\text{e}{-}01 \\ -6.51294820620799 69\text{e}{-}01 \\ 1.21427142372241 28\text{e}{-}04 \\ -1.22593169029294 82\text{e}{-}03 \\ 6.33034757323524 43\text{e}{-}01 \end{bmatrix}^T. \tag{B.12}$$

Finally, the average reprojection error has been found to be: 3.7273295810922097e−01.