Compensation for latency in XR offloaded tasks using pose prediction

Bálint Péter and Yas Yazdanian

DEPARTMENT OF DESIGN SCIENCES
FACULTY OF ENGINEERING LTH | LUND UNIVERSITY
2024

MASTER THESIS

Compensation for Latency in XR Offloaded Tasks using pose prediction

Bálint Péter and Yas Yazdanian

LUND UNIVERSITY

Compensation for Latency in XR Offloaded Tasks using pose prediction

Published by
Department of Design Sciences
Faculty of Enginnering LTH, Lund University
Box 118
SE-221 00 LUND
Sweden

Subject: Degree Project in Virtual Reality and Augmented Reality (MAMM15)
Department of Design Sciences
Supervisor(s): Günter Alce, gunter.alce@design.lth.se and
Marcus Valtonen Örnhag, marcus.valtonen.ornhag@ericsson.com and
Leire Amenabar, leire.amenabar@ericsson.com and
Tobias Widmark, tobias.widmark@ericsson.com
Examiner: Joakim Eriksson, joakim.eriksson@design.lth.se

Abstract

As Augmented Reality (AR) and Mixed Reality (MR) glasses continue to advance, becoming more compact and user-friendly, certain computationally demanding tasks are being offloaded to edge networks or the cloud. This shift, while enhancing the capabilities of AR/MR glasses, will introduce a new challenge—latency.

Latency occurs when there is a need to transmit data to a remote processing unit, perform tasks, and then send the processed information back to the device for rendering. Poor network conditions prevent the timely rendering of content on AR/MR glasses which negatively impacts the user experience and in worst case could lead to Virtual Reality (VR) sickness.

In this thesis, we will focus on minimizing the perceived latency to enhance the user experience in AR/MR applications. Our primary approach involves using head movement tracking with available sensors to minimize latency and synchronize content with user movements, leading to a seamless and enjoyable immersive experience.

Keywords: compensation, latency, XR, offloading, pose prediction

Sammanfattning

I takt med att glasögon för Augmented Reality (AR) och Mixed Reality (MR) blir tunnare och mer användarvänliga är det nödvändigt att utföra vissa beräkningsintensiva uppgifter på externa processorer, däribland edge-nätverk eller molnet. Denna förflyttning av beräkningsresurser, som förbättrar kapaciteten hos AR/MR-glasögon, introducerar en ny utmaning—latens.

Latens uppstår när det finns ett behov av att överföra data till en extern processor, utföra beräkningar och sedan skicka tillbaka den bearbetade informationen till enheten för rendering. Ibland kan dåliga nätverksförhållanden förhindra att innehåll renderas i tid på AR/MR-glasögon, vilket resulterar i en negativ påverkan av användarupplevelsen och i värsta fall bidrar till *Virtual Reality (VR) sickness*.

I denna uppsats fokuserar vi på att motverka latens för att förbättra användarupplevelsen i AR/MR-applikationer. Detta görs främst genom att använda sensorinformation från användarens huvudrörelser för att beräkna skillnaden i position mellan platsen där bilden tagits och renderingsplatsen. Denna metod kan minimera latensen och synkronisera innehållet med användarens rörelser, vilket leder till en bättre upplevelse.

Nyckelord: kompensation, latens, XR, avlastning, positionsprediktering

Acknowledgements

We would like to express our gratitude to our supervisors. From Ericsson, we extend our heartfelt thanks to Marcus Valtonen Örnhag for his immense support at any time of the day, as well as to Leire Amenabar and Tobias Widmark. From Lund University, we are deeply grateful to Günter Alce for his invaluable feedback. Special thanks go to Martin Ek for his selfless help.

Our gratitude also extends to friends and family who supported us during our work: Amir Rashidinia and Rose Rouhani, as well as Gábor Vagyóczki, Aimée Niau Lacordaire, and Pauline Zaragoza.

Lund, May 2024
Bálint Péter and Yas Yazdanian

Table of contents

List of acronyms and abbreviations 9

1. Introduction 11
  - 1.1 Background 11
  - 1.2 Goals 12
    - 1.2.1 Research Questions 13
  - 1.3 Contribution to United Nations Sustainable Development Goals 14
  - 1.4 Related Work 14

2. Theoretical background 16
  - 2.1 Augmented Reality 16
    - 2.1.1 Optical See-Through 16
    - 2.1.2 Video Pass-Through 16
  - 2.2 Virtual Reality Sickness and Latency 17
  - 2.3 Tracking Methods in XR HMDs 18
  - 2.4 Linear Interpolation 19
  - 2.5 The Pinhole Camera Model 19
  - 2.6 The Homography Matrix 20
    - 2.6.1 Homography from Point Correspondences 21
      - 2.6.1.1 Direct Linear Transform (DLT) Algorithm 22
    - 2.6.2 Homography from Camera Displacement 22
      - 2.6.2.1 Computing the Normal Vector n and Distance d 23
  - 2.7 Camera Calibration 23
  - 2.8 Calibrating Between Reality and a Virtual Environment (in Unity) 25

3. Method 30
  - 3.1 Phase One: Exploration and Preparation 30
    - 3.1.1 Literature Study 30
    - 3.1.2 Hardware Setup and Test Environment 31
      - 3.1.2.1 Raspberry Pi Setup for Optical See-Through 32
    - 3.1.3 Creating a Latency Simulator 33
    - 3.1.4 Measuring Baseline Noise and Latency 33
  - 3.2 Phase Two: Implementation 36

3.2.1 Developing the Server 36
3.2.2 Developing the Client 37
3.2.3 Creating a Predictor 37
3.2.4 Running the Prediction On-Device 39
3.2.5 Developing for the Varjo XR3 41
3.3 Phase Three: Evaluation 42
4. Results 46
5. Discussion 48
5.1 Discussion of Results 48
5.2 Evaluating Our Process 48
5.3 Limitations 49
5.4 Addressing Research Questions 50
5.5 Future Work 51
6. Conclusion 54
References 55

List of acronyms and abbreviations

AI	artificial intelligence
AR	augmented reality
FPS	frames per second
HMD	head-mounted display
IMU	inertial measurement unit
LERP	linear interpolation
MR	mixed reality
SDG	sustainable development goal
VOR	vestibule ocular reflex
VR	virtual reality
WebRTC	web real-time communication
XR	extended reality (umbrella term for augmented, mixed, and virtual reality)

1 Introduction

This chapter outlines the scope, objectives, and significance of the thesis. It describes our research goals and elaborates on the contribution of this study towards the United Nations Sustainable Development Goals. Lastly, we finish the chapter by presenting previous studies relevant to our thesis.

1.1 Background

In recent years, Augmented Reality (AR) and Mixed Reality (MR) have been increasingly popular amongst the public, transforming how we interact with digital content. These technologies offer immersive experiences that blend virtual elements with the real world, through the use of Head-Mounted Display (HMD) devices [13]. HMDs are equipped with two display modules for displaying visual content directly to the user's eyes. Additionally, HMDs have a tracking system mounted on them which is responsible for tracking the position and orientation of the user's head in a three-dimensional space. The information gathered from the tracking data is then used to ensure that the virtual content aligns accurately with the user's perspective [3].

As the demand for more user-friendly HMDs grows, there is a simultaneous push towards making these devices lighter and more comfortable to wear [9]. This shift towards lighter, more wearable devices, means that certain computationally intensive tasks has to be offloaded to a remote processing unit. Lighter and more user-friendly HMDs can be made while ensuring they remain powerful and efficient tools for users.

The Web Real-Time Communication (WebRTC) protocol is specifically designed for the efficient offloading of video streams and can be effectively used for this purpose. In our case, the offloading process involves a Unity client, which acts as the interface for the AR/MR application, transmitting video streams to a dedicated server. Upon receiving the stream, the server processes the content and adds the necessary metadata, such as timestamps and pose data, before sending it back to the client. Figure 1.1.1 illustrates the WebRTC protocol's role in facilitating this offloading process.

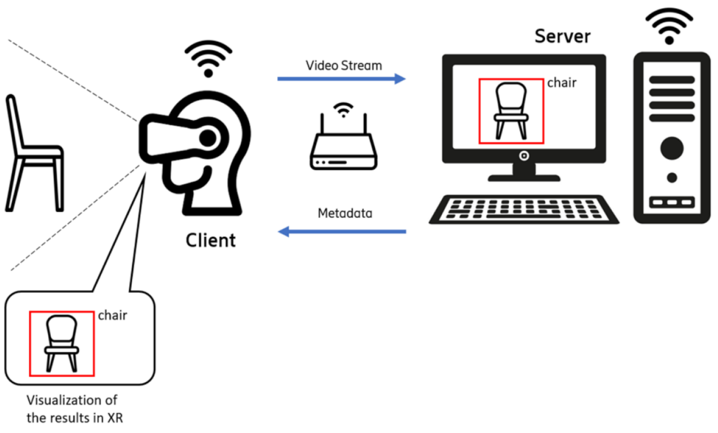


Figure 1.1.1 Architecture of an Offloaded Augmented Reality Application.

This offloading process, while solving computational challenges, introduces latency as a consequence. Latency, in this context, refers to the delay from the moment data is sent from the HMD to the server until the processed content is returned and rendered on the device [31]. This delay can significantly affect the user experience, especially in real-time applications where immediate response to user actions is essential. In AR applications latency can disrupt the seamless integration of virtual and real worlds, potentially leading to disorientation or a less convincing immersive experience [24]. Additionally, latency can cause VR sickness, a topic we discuss in more detail in Section 2.2.

1.2 Goals

The main objective of our thesis is to devise a method for compensating the latency introduced by offloading tasks in extended reality (XR) environments, particularly when real-time responsiveness is crucial. We begin our exploration with a practical example to illustrate our main objective:

Imagine a scenario where a user in an AR setup is observing a cat through their HMD, as depicted in Figure 1.2.1. At the initial moment t_0 the image is transmitted to a remote server which processes the visual data by performming semantic segmentation. The server identifies and spatially locates the cat within the scene as it appeared at time t_0. However, during the time it takes for this information to be processed and sent back to the user, the user's perspective shifts from $\boldsymbol{P_1}$ to $\boldsymbol{P_2}$, altering the image coordinate system due to the change in the user's viewpoint.

Our thesis addresses the challenge of updating the returned data to align it with the user's new perspective. We aim to apply a transformation, specifically a projective transformation (also known as homography), to adjust the virtual content. This transformation recalculates the positions of known objects to reflect changes in the user's orientation and position, effectively adapting the virtual content to correspond with the user's current view.

In practice, we focus on implementing a predictive homography-based approach, utilizing the user's head movements to estimate future states of the environment. This process is detailed in Chapter 4, where we discuss the use of projective transformations to adapt previously known positions to their new projected locations in the user's field of view.

Ultimately, our goal is to utilize these transformations to ensure that despite any movement in the real world, the virtual content remains consistently positioned and oriented relative to the user's new perspective. This approach aims to reduce the perceived latency and improve the responsiveness of the XR system, enhancing the user experience by maintaining a seamless integration of virtual and real-world elements.

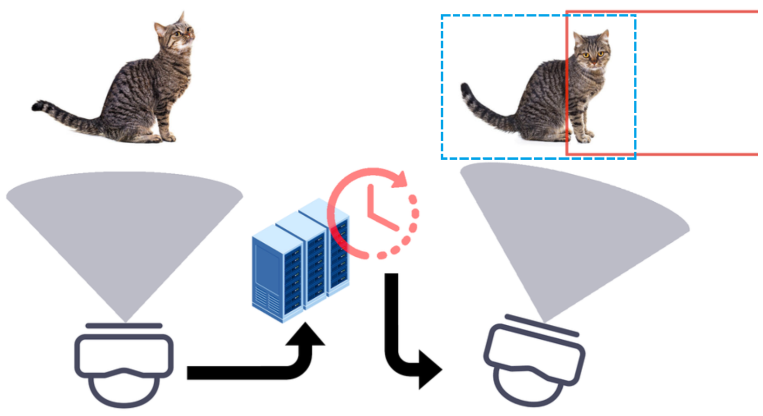


Figure 1.2.1 Example of semantic segmentation used in augmented reality.

1.2.1 Research Questions

1. How can an algorithm reduce perceived latency in offloaded AR/MR applications by compensating for pose differences using sensor data?
    - For what types of motions and applications is this algorithm effective?
2. How can calibration between the physical and virtual worlds be achieved for accurate pose-based prediction?

1.3 Contribution to United Nations Sustainable Development Goals

This research contributes to the United Nations Sustainable Development Goal (SDG) 9 (Industry, Innovation, and Infrastructure) by advancing Augmented Reality (AR) and Mixed Reality (MR) technologies. Through developing innovative solutions for reducing latency, this thesis supports the enhancement of technological infrastructure and fosters innovation. This aligns with SDG 9's objectives to build resilient infrastructure, promote inclusive and sustainable industrialization, and encourage innovation.

Offloading also enables the use of simpler chip architectures in consumer devices, enhancing power efficiency. This aligns with SDG 12, responsible consumption and production, by reducing the need for complex components. It also supports Goal 13, climate action, due to the potential positive environmental impact from lower energy consumption. Additionally, streamlined chip designs can lead to reduced e-waste and more sustainable production practices.

9 INDUSTRY, INNOVATION AND INFRASTRUCTURE
12 RESPONSIBLE CONSUMPTION AND PRODUCTION
13 CLIMATE ACTION

(a) (b) (c)

Figure 1.3.1 UN development goals.

1.4 Related Work

Prior studies have explored head motion prediction as a strategy to address on-device latency in initial extended reality (XR) systems, even preceding the development of offloaded XR technologies. The mathematical frameworks employed for these predictions have historical applications beyond XR, including the forecasting of financial trends and weather patterns [17].

Ongoing research in this domain is increasingly incorporating AI to enhance prediction accuracy. Currently, certain AI models have demonstrated improved performance over purely mathematical methods, such as the Kálmán Filter and Double Exponential Smoothing algorithm [31].

Cloud computing has been, and remains to be, an active research area integrating applications from big data analytics to cloud gaming [7]. Recent advancements in offloaded rendering include workload distribution between clients and servers, and effective frame caching strategies, like Flashback by Microsoft [5]. Additionally, innovations like Furion [35] optimize Wi-Fi power efficiency, while Microsoft's Kahawai [11] introduces a method for offloading only a part of GPU processes.

Specifically, offloading VR has also been explored in detail. The study of VR offloading includes various techniques, which aim to enhance rendering efficiency and reduce network demands. Notably, several approaches focus on the user's gaze to allocate more resources to the viewed area, e.g., complemented by foveated rendering, which utilizes eye tracking to decrease graphical fidelity in peripheral vision without effecting user experience. Moreover, some strategies involve encoding selected low-impact frames at lower bit rates to ease network requirements [16].

A study on homography-based loss functions for camera pose regression demonstrates how homography can be effectively utilized for accurate pose estimation in augmented reality applications. This approach involves computing homographies for a set of virtual parallel planes to approximate reprojection error while avoiding some of its drawbacks. The homography-based method offers competitive accuracy and high numerical stability, making it suitable for real-time AR applications. By quantifying error as the difference between the identity matrix and the homography induced by planes between the ground truth and estimated poses, this technique contributes to reduced latency and improved user experience [4].

2 Theoretical background

This chapter provides comprehensive overview of the concepts required to fully grasp this thesis.

2.1 Augmented Reality

Augmented Reality uses a see-through HMD to overlay virtual objects into the real world. Unlike virtual reality where the whole environment displayed to the user is virtual, AR combines the real and virtual world together. (This same description can apply to MR. However, the distinctions between mixed reality and augmented reality are often unclear and subject to differing opinions. Therefore, for the purposes of this thesis, we will refer to this type of technology as AR.) This integration is accomplished via optical or video-see through HMDs [2]. Therefore, we have categorized HMDs into two sections; optical see-through HMDs and video pass-through HMDs.

2.1.1 Optical See-Through

In optical see-through HMDs, partially transmissive optical combiners are placed in front of the user's eyes. These combiners allow the user to see the real world directly through them, which means the real environment remains visible even when the HMD is turned off. Additionally, these combiners are also partially reflective, enabling the projection of virtual images from the head-mounted monitor onto the combiner, where they are then seen by the user. This setup creates a seamless integration of virtual objects into the user's view of the real world [3]. Figure 2.1.1 illustrates the components and functionality of an optical see-through HMD.

2.1.2 Video Pass-Through

In video pass-through HMDs, the headset completely covers the eyes, unlike optical see-through HMDs. This means that if the device is turned off, the user cannot see the real world through the HMD. The view of the real world is provided by a combination of a closed view HMD with one or two head mounted video cameras. The video from these cameras will be combined with the created virtual content by the scene generator and sent to the monitor for the user to see [3]. Figure 2.1.2 illustrates the components and functionality of a video pass-through HMD.

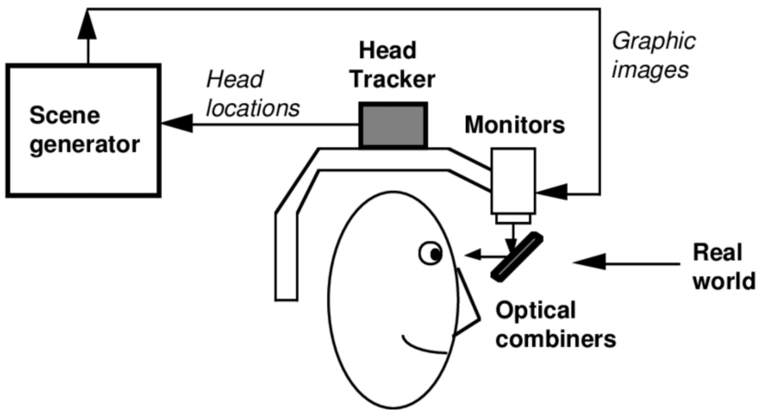


Figure 2.1.1 Conceptual diagram of an optical see-through HMD [3].

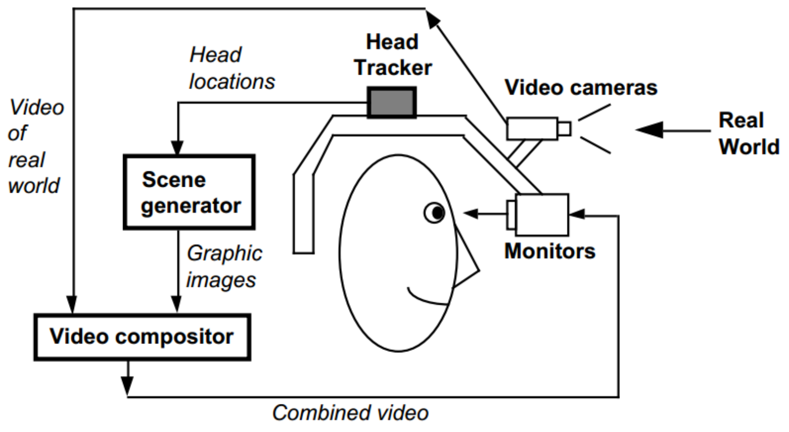


Figure 2.1.2 Conceptual diagram of a video pass-through HMD [3].

2.2 Virtual Reality Sickness and Latency

Unlike AR, which overlays virtual objects onto the real world, VR creates immersive, computer-generated environments using video pass-through HMDs.

Latency is a critical factor in VR environments as it can significantly contribute to VR sickness, also known as cybersickness. VR sickness occurs when there is a mismatch between the sensory inputs received by the eyes and the vestibular system in the inner ear, which helps control balance and eye movements [8]. The Vestibulo-Ocular Reflex (VOR) is a mechanism that stabilizes vision during head movements by producing eye movements in the opposite direction of head movement, allowing the gaze to remain fixed on an object. When latency in the system causes a delay in updating the visual scene in response to head movements, the VOR cannot function correctly, leading to sensory conflict and VR sickness symptoms such as nausea, dizziness, and disorientation [15]. The latency of the VOR varies depending on the type of head movement: for rotational movements, the latency is approximately 4-17 milliseconds, while for translational movements, it ranges from 20-62 milliseconds [10]. Therefore, minimizing system latency is crucial to enhance the comfort and usability of VR systems.

2.3 Tracking Methods in XR HMDs

Tracking in XR devices is crucial for ensuring accurate alignment between the virtual and real worlds. Various methods are employed to achieve this, including the use of Inertial Measurement Units (IMUs), inside-out tracking, outside-in tracking, and image-based tracking.

IMUs, which consist of accelerometers and gyroscopes, are commonly integrated into HMDs to track the movement and orientation of the user's head. These sensors provide high-frequency data that is essential for smooth and responsive tracking, although they may suffer from drift over time without correction [25].

Inside-out tracking utilizes cameras and sensors mounted on the HMD itself to observe the surrounding environment. By recognizing and tracking features or markers within the user's vicinity, this method offers a self-contained solution that is convenient and scalable. It negates the need for external tracking systems, making it ideal for portable and standalone HMDs.

Outside-in tracking, on the other hand, relies on external cameras or sensors positioned around the user. These devices track the HMD and, sometimes, handheld controllers, offering potentially higher accuracy and stability since the external references are fixed in the environment. This method is often used in high-end VR systems where precise tracking is paramount.

Image-based tracking leverages computer vision techniques to process images captured by the HMD's cameras. This method can identify and track physical objects or markers within the environment, such as Valve's lighthouse trackers [27], contributing to more robust and detailed positional tracking. Image-based tracking is often combined with other methods, such as IMUs, to enhance overall tracking performance and reduce latency.

By integrating these various tracking methods, XR HMDs can deliver an immersive and responsive user experience, crucial for applications ranging from gaming to professional simulations.

2.4 Linear Interpolation

Linear Interpolation (LERP) is a method used to estimate unknown values that fall within the range of two known values. In mathematical terms, if we have two points (x_0, y_0) and (x_1, y_1), the linear interpolation formula is

$$y = y_0 + (y_1 - y_0)\frac{x - x_0}{x_1 - x_0}, \tag{2.1}$$

where x is the point at which we want to estimate the value of y. The formula essentially constructs a straight line between the two known points and uses this line to find the estimated value.

LERP is frequently used to generate intermediate values, smooth transitions, or fill in gaps in data. For example, when animating a moving object, linear interpolation can be used to determine the object's position at any given time between two keyframes [22].

2.5 The Pinhole Camera Model

The pinhole camera model describes the mathematics of transforming a world point into an image point. Coupled with a distortion model that characterizes the deviations from the pinhole model, it is possible to accurately represent most cameras using this method [18].

The pinhole camera model is a simple representation of how a camera captures an image. It assumes that light rays pass through a single point (the pinhole) and form an image on the opposite side. In this model, the relationship between a 3D point $\boldsymbol{X} = [X, Y, Z]^T$ in the world coordinates and its corresponding 2D point $\boldsymbol{x} = [x, y]^T$ in the image coordinates is given by

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \boldsymbol{K} \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{2.2}$$

where $\boldsymbol{K}$ is the intrinsic camera matrix, $\boldsymbol{R}$ is the rotation matrix, and $\boldsymbol{t}$ is the translation vector.

The intrinsic camera matrix $\boldsymbol{K}$ is defined as

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & -u_0 \\ 0 & f & -v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.3}$$

where f is the focal length and (u_0, v_0) are the coordinates of the principal point.

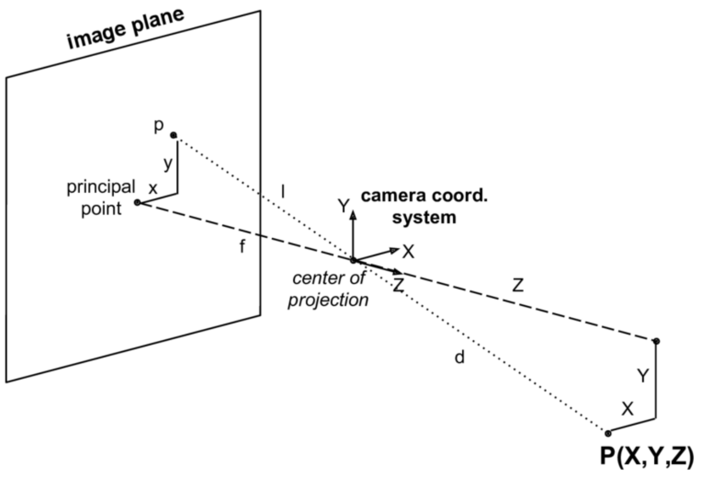


Figure 2.5.1 Pinhole camera model [28].

2.6 The Homography Matrix

The homography matrix is a transformation that maps points from one image plane to another. Homographies are also referred to as projective transformations. Figure 2.6.1 considers a planar surface in the physical world, denoted as π, with two cameras positioned at $\boldsymbol{C}$ and $\boldsymbol{C}'$ capturing this plane. A point on this plane, represented as $\boldsymbol{x}_\pi$, is projected onto the image planes of both cameras, resulting in the image points $\boldsymbol{x}$ and $\boldsymbol{x}'$, respectively. The homography matrix $\boldsymbol{H}$ encapsulates the transformation from the view of camera $\boldsymbol{C}$ to that of camera $\boldsymbol{C}'$, mapping the point $\boldsymbol{x}$ in the first image to the point $\boldsymbol{x}'$ in the second image as expressed in the equation $\boldsymbol{x}' \sim \boldsymbol{H}\boldsymbol{x}$ [14]. Here, the symbol $\sim$ denotes equality up to a scale factor.

The homography $\boldsymbol{H}$ is a 3×3 matrix, which, despite its size, possesses 8 degrees of freedom (DoF) due to global scale ambiguity.

Given a point (x, y) in one image and its corresponding point (x', y') in another image, the relationship between these points through the homography matrix $\boldsymbol{H}$ can be expressed as

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{2.4}$$

where s is a scaler factor, and $\boldsymbol{H}$ is the homography matrix with elements h_{ij}.

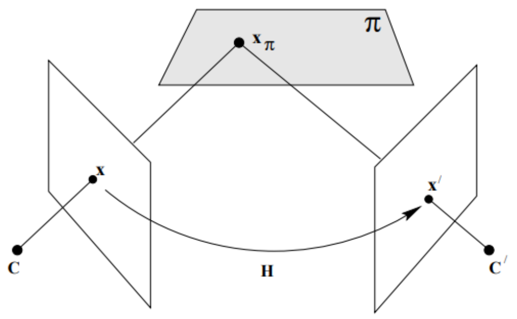


Figure 2.6.1 Planar homography [14].

2.6.1 Homography from Point Correspondences

The process of computing the homography matrix $\boldsymbol{H}$ from point correspondences involves utilizing the geometric relationship between pairs of points on a plane in one image and their counterparts in another image. Given a set of point correspondences, where $\boldsymbol{p}_i$ and $\boldsymbol{p}_i'$ represent corresponding points in two different images, the homography matrix $\boldsymbol{H}$ can be computed that best maps all points from the first image to their locations in the second image.

To compute the homography, we consider each point correspondence $\boldsymbol{p}_i$ and $\boldsymbol{p}_i'$ as homogeneous coordinates. For a set of at least four point correspondences, the homography $\boldsymbol{H}$ satisfies the equation

$$\boldsymbol{p}_i' \sim \boldsymbol{H}\boldsymbol{p}_i. \tag{2.5}$$

Expanding the equation and applying the cross product, we obtain

$$\boldsymbol{p}_i' \times (\boldsymbol{H}\boldsymbol{p}_i) = \boldsymbol{0}, \tag{2.6}$$

which ensures that $\boldsymbol{p}_i'$ and $\boldsymbol{H}\boldsymbol{p}_i$ are collinear. This equation can be expanded into two independent equations for each point correspondence, leading to a system of linear equations that can be represented in matrix form as

$$\boldsymbol{A}\boldsymbol{h} = \boldsymbol{0}, \tag{2.7}$$

where

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & -x_i & -y_i & -1 & y_i'x_i & y_i'y_i & y_i' \\ x_i & y_i & 1 & 0 & 0 & 0 & -x_i'x_i & -x_i'y_i & -x_i' \end{bmatrix}, \tag{2.8}$$

is constructed from the point correspondences in the way shown in (2.8), $\boldsymbol{h}$ is the vectorized form of the homography matrix $\boldsymbol{H}$, $\boldsymbol{p}_i = \begin{bmatrix} x_i & y_i & 1 \end{bmatrix}^T$ and $\boldsymbol{p}_i' = \begin{bmatrix} x_i' & y_i' & 1 \end{bmatrix}^T$.

2.6.1.1 Direct Linear Transform (DLT) Algorithm

The Direct Linear Transform (DLT) algorithm provides a method to solve for $\boldsymbol{h}$ by constructing the matrix $\boldsymbol{A}$ from all point correspondences, as shown in (2.9), and then solving the homogeneous system of equations,

$$\boldsymbol{A} = \begin{bmatrix} 0 & 0 & 0 & -x_1 & -y_1 & -1 & y'_1x_1 & y'_1y_1 & y'_1 \\ x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & -x_2 & -y_2 & -1 & y'_2x_2 & y'_2y_2 & y'_2 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 & -x'_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & -x_N & -y_N & -1 & y'_Nx_N & y'_Ny_N & y'_N \\ x_N & y_N & 1 & 0 & 0 & 0 & -x'_Nx_N & -x'_Ny_N & -x'_N \end{bmatrix}. \tag{2.9}$$

For $N = 4$ distinct point correspondences (non-degenerate configurations), there exists a solution to $\boldsymbol{Ah} = \boldsymbol{0}$; however, in the presence of noise, one often considers $N > 4$ points. In general, no solution exists but instead one seeks to minimize $\min_h ||\boldsymbol{Ah}||_F$, where $||\cdot||_F$ is the Frobenius norm. The solution to this problem is given by the eigenvector of $\boldsymbol{A}^T\boldsymbol{A}$ corresponding to the smallest eigenvalue, or equivalently, the singular vector of $\boldsymbol{A}$ corresponding to the smallest singular value obtained through Singular Value Decomposition (SVD) [14].

The vector $\boldsymbol{h}$ is reshaped into the $3x3$ homography matrix $\boldsymbol{H}$. As mentioned before $\boldsymbol{H}$ is determined up to a scale factor; hence, it is common practice to normalize $\boldsymbol{H}$ such that $h_{33} = 1$.

2.6.2 Homography from Camera Displacement

To understand the homography that maps points from the first camera frame to the second, let's consider the changes in poses between two camera positions, $\boldsymbol{P}_1 = [\boldsymbol{R_1}\,\boldsymbol{t_1}]$ and $\boldsymbol{P}_2 = [\boldsymbol{R_2}\,\boldsymbol{t_2}]$. The homography is computed using the rotation and translation changes between these two camera frames. The equation for computing the homography from camera displacement is given by

$$\boldsymbol{H}_1^2 = \boldsymbol{R}_1^2 + \frac{\boldsymbol{t}_1^2\boldsymbol{n}^T}{d}, \tag{2.10}$$

where $\boldsymbol{H}_1^2$ is the homography that maps the points in the first camera frame to the points in the second camera frame [23]. Here, $\boldsymbol{R}_1^2$ represents the 3×3 rotation matrix that defines the rotation from camera frame 1 to camera frame 2, $\boldsymbol{t}_1^2$ is the translation vector representing the translation between the two camera frames,$\boldsymbol{n}$ is the normal to the plane, d is the distance from the plane to the camera frame.

Consider the following poses: Pose $\boldsymbol{P}_1 = [\boldsymbol{R}_1\, \boldsymbol{t}_1]$ represents the rotation $\boldsymbol{R}_1$ and translation $\boldsymbol{t}_1$ of the first camera, pose $\boldsymbol{P}_2 = [\boldsymbol{R}_2\, \boldsymbol{t}_2]$ represents the rotation $\boldsymbol{R}_2$ and translation $\boldsymbol{t}_2$ of the second camera.

The relative rotation $\boldsymbol{R}_1^2$ and translation $\boldsymbol{t}_1^2$ from camera 1 to camera 2 can then be calculated as

$$\boldsymbol{R}_1^2 = \boldsymbol{R}_2 \boldsymbol{R}_1^T, \tag{2.11}$$

and

$$\boldsymbol{t}_1^2 = \boldsymbol{R}_2(-\boldsymbol{R}_1^T \boldsymbol{t}_1) + \boldsymbol{t}_2, \tag{2.12}$$

respectively.

2.6.2.1 Computing the Normal Vector *n* and Distance *d*

The normal vector $\boldsymbol{n}$ of the plane represents the orientation of the plane in 3D space. It can be derived from the rotation matrices. If $\boldsymbol{R}_1$ and $\boldsymbol{R}_2$ are the rotation matrices corresponding to the initial and displaced camera positions, then

$$\boldsymbol{n} = \frac{\boldsymbol{R}_1 \boldsymbol{n}_0}{\|\boldsymbol{R}_1 \boldsymbol{n}_0\|}, \tag{2.13}$$

where $\boldsymbol{n}_0$ is the initial normal vector of the plane. This is a vector that indicates the orientation of the plane before any camera movements. The normal vector $\boldsymbol{n}$ is thus transformed by the initial rotation matrix $\boldsymbol{R}_1$ and normalized.

The distance d from the camera to the plane is computed based on the position of the plane relative to the camera. Given a point $\boldsymbol{X}_0$ on the plane, the distance d can be calculated as

$$d = \frac{\boldsymbol{n} \cdot (\boldsymbol{R}_1 \boldsymbol{X}_0 + \boldsymbol{t}_1)}{\|\boldsymbol{n}\|}. \tag{2.14}$$

This formula ensures that d represents the perpendicular distance from the camera center to the plane along the normal vector $\boldsymbol{n}$.

2.7 Camera Calibration

Camera calibration is a fundamental step in computer vision and photogrammetry. It involves estimating the parameters of the camera that are necessary to capture images in a way that allows for accurate measurement and analysis. This process typically includes determining the intrinsic matrix $\boldsymbol{K}$ and correcting for lens distortions to produce undistorted images.

To calculate $\boldsymbol{K}$ and the distortion parameters, one may use Zhang's method [34], which is homography-based. In this method, one captures multiple images of a

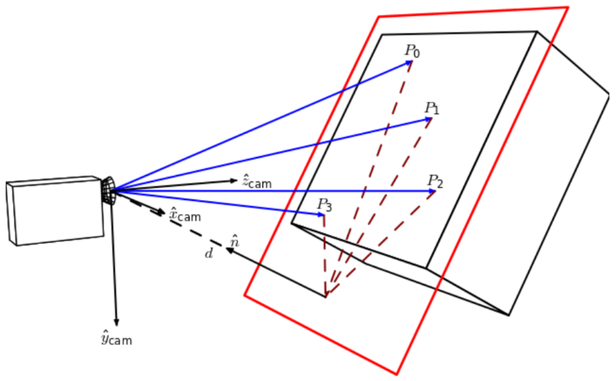


Figure 2.6.2 Homography from Camera Displacement [23].

known calibration object, such as a checkerboard pattern. The calibration process involves the following steps:

1. Image Acquisition: Capture multiple images of the calibration object from different angles and distances.

2. Feature Detection: Detect feature points (corners of the checkerboard squares) in the images.

3. Correspondence Matching: Establish correspondences between the detected feature points in the images and the known coordinates of the points on the calibration object.

4. Optimization: Use optimization techniques to minimize the reprojection error, which is the difference between the observed feature points in the images and the projected points from the 3D model using the estimated intrinsic matrix.

Most camera lenses introduce some degree of distortion to the captured images. The most common types of lens distortion are radial and tangential distortion. Radial Distortion: This type of distortion causes straight lines to appear curved. It is characterized by the radial distortion coefficients k_1, k_2, and k_3.

Tangential Distortion: This type of distortion occurs when the lens and the image plane are not perfectly parallel. It is characterized by the tangential distortion coefficients p_1 and p_2.

The distortion in imaging can be effectively modeled using the Brown–Conrady distortion model [6], as described by the following equations

$$\begin{aligned} x_{\text{distorted}} &= x\left(1+k_1r^2+k_2r^4+k_3r^6\right)+2p_1xy+p_2\left(r^2+2x^2\right), \\ y_{\text{distorted}} &= y\left(1+k_1r^2+k_2r^4+k_3r^6\right)+p_1\left(r^2+2y^2\right)+2p_2xy, \end{aligned} \tag{2.15}$$

where $r^2 = x^2 + y^2$.

Once the distortion coefficients have been estimated, we can correct the distorted images to produce undistorted images. The undistortion process involves the following steps:

1. Compute the Undistortion Map: Using the distortion coefficients and the intrinsic matrix $\boldsymbol{K}$, compute the mapping from distorted pixel coordinates to undistorted pixel coordinates.
2. Apply the Undistortion Map: Use the undistortion map to remap the pixels in the distorted image to their corrected positions in the undistorted image.

Camera calibration is a critical process in computer vision that enables accurate measurement and analysis of images. By calculating the intrinsic matrix $\boldsymbol{K}$ and correcting for lens distortions, we can produce undistorted images that are suitable for further processing and analysis. This calibration is essential because it ensures the validity of the pinhole camera model.

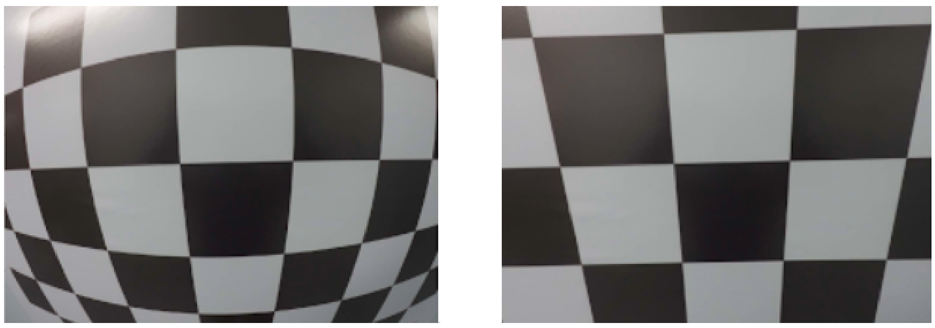


Figure 2.7.1 Original image (on the left) compared to undistorted image (on the right), after correcting lens distortion [21].

2.8 Calibrating Between Reality and a Virtual Environment (in Unity)

Consider a pose $\boldsymbol{P}_j = [\boldsymbol{R}_j \mid \boldsymbol{t}_j]$ w.r.t. the Unity coordinate system for view j. Furthermore, in the field-of-view there are N visible markers $\{\boldsymbol{p}_i\}_{i=1}^{N}$ corresponding to known 3D points $\{\boldsymbol{X}_i\}_{i=1}^N$. The 3D points are anchored to the Unity coordinate system and manually aligned with the physical markers. The *normalized* coordinates $\hat{\boldsymbol{x}}_i^{(j)}$ (w.r.t. the Unity coordinate system) are therefore given by $\hat{\boldsymbol{x}}_i^{(j)} = \boldsymbol{P}_j \boldsymbol{X}_i$ for point i in view j. This is the space which the Unity application and the pose data from the HMDs adhere to, hence is the space in which we must define our sought projective transformation. In order to do so, we must calibrate our camera coordinate space accordingly. To this end, we seek a calibration matrix $\boldsymbol{K}$

$$\boldsymbol{K} = \begin{bmatrix} f & 0 & -u_0 \\ 0 & f & -v_0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.16}$$

where f is a scale factor (or focal length) and (u_0, v_0) is the principal point, such that $\boldsymbol{p}_i^{(j)} = \boldsymbol{K}\hat{\boldsymbol{x}}_i^{(j)}$. Let $\hat{\boldsymbol{x}}_i^{(j)} = [x_i^{(j)}, y_i^{(j)}, z_i^{(j)}] \in \mathbb{P}^2$, then

$$\boldsymbol{p}_i^{(j)} = \boldsymbol{K}\hat{\boldsymbol{x}}_i^{(j)} = \begin{bmatrix} f x_i^{(j)} - u_0 z_i^{(j)} \\ f y_i^{(j)} - v_0 z_i^{(j)} \\ z_i^{(j)} \end{bmatrix}. \tag{2.17}$$

Let $(\hat{u}_i^{(j)}, \hat{v}_i^{(j)})$ denote the real-valued representation of $\boldsymbol{p}_i^{(j)}$, then

$$\begin{bmatrix} \hat{u}_i^{(j)} \\ \hat{v}_i^{(j)} \end{bmatrix} = \begin{bmatrix} f\bar{x}_i^{(j)} - u_0 \\ f\bar{y}_i^{(j)} - v_0 \end{bmatrix}, \tag{2.18}$$

where $(\bar{x}_i^{(j)}, \bar{y}_i^{(j)}) := (x_i^{(j)}/z_i^{(j)}, y_i^{(j)}/z_i^{(j)})$ is simply the real-valued representation of $\hat{\boldsymbol{x}}_i^{(j)}$.

From (2.18) we seek f, u_0, and v_0, while the other values are known. In the remaining parts of this section, we will devise a robust calibration scheme to obtain the unknown parameters, using multiple views and multiple 3D points this calibration scheme represents a novel contribution of our work, providing a new method for accurate alignment between the virtual and physical environments.

Theorem 1 (Optimal calibration)**.** *Given 2D correspondences of M known 3D points in N different views, the optimal choice of parameters f, u_0, v_0 of* (2.16) *in the least-squares sense is given by*

$$f = \frac{MN\left(\bar{\boldsymbol{x}}^T\hat{\boldsymbol{u}} + \bar{\boldsymbol{y}}^T\hat{\boldsymbol{v}}\right) - s(\hat{\boldsymbol{u}})s(\bar{\boldsymbol{x}}) - s(\hat{\boldsymbol{v}})s(\bar{\boldsymbol{y}})}{MN\left(\|\bar{\boldsymbol{x}}\|^2 + \|\bar{\boldsymbol{y}}\|^2\right) - s(\bar{\boldsymbol{x}})^2 - s(\bar{\boldsymbol{y}})^2}, \tag{2.19}$$

and

$$u_0 = \frac{MN\Big(\big(\bar{\boldsymbol{x}}^T\hat{\boldsymbol{u}} + \bar{\boldsymbol{y}}^T\hat{\boldsymbol{v}}\big)\,s(\bar{\boldsymbol{x}}) - \big(\|\bar{\boldsymbol{x}}\|^2 + \|\bar{\boldsymbol{y}}\|^2\big)\,s(\hat{\boldsymbol{u}})\Big) + s(\hat{\boldsymbol{u}})s(\bar{\boldsymbol{y}})^2 - s(\hat{\boldsymbol{v}})s(\bar{\boldsymbol{x}})s(\bar{\boldsymbol{y}})}{MN\big(MN\,(\|\bar{\boldsymbol{x}}\|^2 + \|\bar{\boldsymbol{y}}\|^2) - s(\bar{\boldsymbol{x}})^2 - s(\bar{\boldsymbol{y}})^2\big)}, \tag{2.20}$$

$$v_0 = \frac{MN\Big(\big(\bar{\boldsymbol{x}}^T\hat{\boldsymbol{u}} + \bar{\boldsymbol{y}}^T\hat{\boldsymbol{v}}\big)\,s(\bar{\boldsymbol{y}}) - \big(\|\bar{\boldsymbol{x}}\|^2 + \|\bar{\boldsymbol{y}}\|^2\big)\,s(\hat{\boldsymbol{v}})\Big) + s(\hat{\boldsymbol{v}})s(\bar{\boldsymbol{x}})^2 - s(\hat{\boldsymbol{u}})s(\bar{\boldsymbol{x}})s(\bar{\boldsymbol{y}})}{MN\big(MN\,(\|\bar{\boldsymbol{x}}\|^2 + \|\bar{\boldsymbol{y}}\|^2) - s(\bar{\boldsymbol{x}})^2 - s(\bar{\boldsymbol{y}})^2\big)}, \tag{2.21}$$

where $s(\cdot)$ denotes the sum of all elements, $\bar{\boldsymbol{x}}$ is the vector of $\bar{x}_i^{(j)}$, $\bar{\boldsymbol{y}}$ is the vector of $\bar{y}_i^{(j)}$, $\hat{\boldsymbol{u}}$ is the vector of $\hat{u}_i^{(j)}$ and $\hat{\boldsymbol{v}}$ is the vector of $\hat{v}_i^{(j)}$

Proof. We seek to solve multiple equations of the form (2.18) simultaneously. Due to noise, the problem is in general overdetermined, hence we seek to minimize the cost function Ψ

$$\min_{f,u_0,v_0} \Psi(f,u_0,v_0) := \min_{f,u_0,v_0} \sum_{i=1}^{M}\sum_{j=1}^{N} \left(f\bar{x}_i^{(j)} - u_0 - \hat{u}_i^{(j)}\right)^2 + \left(f\bar{y}_i^{(j)} - v_0 - \hat{v}_i^{(j)}\right)^2, \tag{2.22}$$

which can be written as $\Psi(f,u_0,v_0) = \|\boldsymbol{\Phi}(f,u_0,v_0)\|_2^2 = \boldsymbol{\Phi}(f,u_0,v_0)^T\boldsymbol{\Phi}(f,u_0,v_0)$, where

$$\boldsymbol{\Phi}(f,u_0,v_0)=\begin{bmatrix} f\bar{x}_1^{(1)}-u_0-\hat{u}_1^{(1)}\\ f\bar{y}_1^{(1)}-v_0-\hat{v}_1^{(1)}\\ \vdots\\ f\bar{x}_M^{(1)}-u_0-\hat{u}_M^{(1)}\\ f\bar{y}_M^{(1)}-v_0-\hat{v}_M^{(1)}\\ \vdots\\ f\bar{x}_1^{(2)}-u_0-\hat{u}_1^{(2)}\\ f\bar{y}_1^{(2)}-v_0-\hat{v}_1^{(2)}\\ \vdots\\ f\bar{x}_M^{(2)}-u_0-\hat{u}_M^{(2)}\\ f\bar{y}_M^{(2)}-v_0-\hat{v}_M^{(2)}\\ \vdots\\ f\bar{x}_1^{(N)}-u_0-\hat{u}_1^{(N)}\\ f\bar{y}_1^{(N)}-v_0-\hat{v}_1^{(N)}\\ \vdots\\ f\bar{x}_M^{(N)}-u_0-\hat{u}_M^{(N)}\\ f\bar{y}_M^{(N)}-v_0-\hat{v}_M^{(N)} \end{bmatrix} = f\underbrace{\begin{bmatrix} \bar{x}_1^{(1)}\\ \bar{y}_1^{(1)}\\ \vdots\\ \bar{x}_M^{(1)}\\ \bar{y}_M^{(1)}\\ \vdots\\ \bar{x}_1^{(2)}\\ \bar{y}_1^{(2)}\\ \vdots\\ \bar{x}_M^{(2)}\\ \bar{y}_M^{(2)}\\ \vdots\\ \bar{x}_1^{(N)}\\ \bar{y}_1^{(N)}\\ \vdots\\ \bar{x}_M^{(N)}\\ \bar{y}_M^{(N)} \end{bmatrix}}_{:=\boldsymbol{m}} -u_0\underbrace{\begin{bmatrix}1\\0\\\vdots\\1\\0\\\vdots\\1\\0\\\vdots\\1\\0\\\vdots\\1\\0\\\vdots\\1\\0\end{bmatrix}}_{:=\boldsymbol{e}_1} -v_0\underbrace{\begin{bmatrix}0\\1\\\vdots\\0\\1\\\vdots\\0\\1\\\vdots\\0\\1\\\vdots\\0\\1\\\vdots\\0\\1\end{bmatrix}}_{:=\boldsymbol{e}_2} - \underbrace{\begin{bmatrix} \hat{u}_1^{(1)}\\ \hat{v}_1^{(1)}\\ \vdots\\ \hat{u}_M^{(1)}\\ \hat{v}_M^{(1)}\\ \vdots\\ \hat{u}_1^{(2)}\\ \hat{v}_1^{(2)}\\ \vdots\\ \hat{u}_M^{(2)}\\ \hat{v}_M^{(2)}\\ \vdots\\ \hat{u}_1^{(N)}\\ \hat{v}_1^{(N)}\\ \vdots\\ \hat{u}_M^{(N)}\\ \hat{v}_M^{(N)} \end{bmatrix}}_{:=\boldsymbol{d}}, \tag{2.23}$$

where we introduce the compact form $\boldsymbol{\Phi}(f,u_0,v_0)=f\boldsymbol{m}-u_0\boldsymbol{e}_1-v_0\boldsymbol{e}_2-\boldsymbol{d}$. It follows that

$$\begin{aligned}\Psi &= (f\boldsymbol{m}-u_0\boldsymbol{e}_1-v_0\boldsymbol{e}_2-\boldsymbol{d})^T(f\boldsymbol{m}-u_0\boldsymbol{e}_13-v_0\boldsymbol{e}_2-\boldsymbol{d})\\ &= f^2\boldsymbol{m}^T\boldsymbol{m}-2fu_0\boldsymbol{m}^T\boldsymbol{e}_1-2fv_0\boldsymbol{m}^T\boldsymbol{e}_2-2f\boldsymbol{m}^T\boldsymbol{d}\\ &\quad+u_0^2\boldsymbol{e}_1^T\boldsymbol{e}_1+2u_0v_0\boldsymbol{e}_1^T\boldsymbol{e}_2+2u_0\boldsymbol{e}_1^T\boldsymbol{d}\\ &\quad+v_0^2\boldsymbol{e}_2^T\boldsymbol{e}_2+2v_0\boldsymbol{e}_2\boldsymbol{d}+\boldsymbol{d}^T\boldsymbol{d}\,.\end{aligned} \tag{2.24}$$

We seek the stationary points; the partial derivatives are given by

$$\frac{\partial\Psi}{\partial f}=2f\boldsymbol{m}^T\boldsymbol{m}-2u_0\boldsymbol{m}^T\boldsymbol{e}_1-2v_0\boldsymbol{m}^T\boldsymbol{e}_2-2\boldsymbol{m}^T\boldsymbol{d}, \tag{2.25}$$

$$\frac{\partial\Psi}{\partial u_0}=-2f\boldsymbol{m}^T\boldsymbol{e}_1+2u_0\boldsymbol{e}_1^T\boldsymbol{e}_1+2v_0\boldsymbol{e}_1^T\boldsymbol{e}_2+2\boldsymbol{e}_1^T\boldsymbol{d}, \tag{2.26}$$

$$\frac{\partial\Psi}{\partial v_0}=-2f\boldsymbol{m}^T\boldsymbol{e}_2+2u_0\boldsymbol{e}_1^T\boldsymbol{e}_2+2v_0\boldsymbol{e}_2^T\boldsymbol{e}_2+2\boldsymbol{e}_2^T\boldsymbol{d}\,. \tag{2.27}$$

Due to the structure of the vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$, we find that $\boldsymbol{e}_1^T\boldsymbol{e}_2 = 0$ and $\boldsymbol{e}_1^T\boldsymbol{e}_1 = \boldsymbol{e}_2^T\boldsymbol{e}_2 = MN$. Therefore, any stationary points fulfil the equation

$$\underbrace{\begin{bmatrix} \boldsymbol{m}^T\boldsymbol{m} & -\boldsymbol{m}^T\boldsymbol{e}_1 & -\boldsymbol{m}^T\boldsymbol{e}_2 \\ -\boldsymbol{m}^T\boldsymbol{e}_1 & MN & 0 \\ -\boldsymbol{m}^T\boldsymbol{e}_2 & 0 & MN \end{bmatrix}}_{:=\boldsymbol{M}} \begin{bmatrix} f \\ u_0 \\ v_0 \end{bmatrix} - \underbrace{\begin{bmatrix} \boldsymbol{m}^T\boldsymbol{d} \\ -\boldsymbol{e}_1^T\boldsymbol{d} \\ -\boldsymbol{e}_2^T\boldsymbol{d} \end{bmatrix}}_{:=\boldsymbol{b}} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} . \tag{2.28}$$

For non-degenerate configurations the matrix $\boldsymbol{M}$ is invertible, hence there exists a unique stationary point. Since (2.22) is strictly convex, this is the global optimum.

We may now solve for the unknowns, which—after simplifications—yields

$$\begin{bmatrix} f \\ u_0 \\ v_0 \end{bmatrix} = \frac{1}{\det \boldsymbol{M}} \begin{bmatrix} MN(MN\boldsymbol{m}^T\boldsymbol{d} - \boldsymbol{e}_1^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{e}_1 - \boldsymbol{e}_2^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{e}_2) \\ \boldsymbol{e}_1^T\boldsymbol{d}(\boldsymbol{m}^T\boldsymbol{e}_2)^2 - \boldsymbol{e}_2^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{e}_1\boldsymbol{m}^T\boldsymbol{e}_2 - MN\boldsymbol{e}_1^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{m} + MN\boldsymbol{m}^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{e}_1 \\ \boldsymbol{e}_2^T\boldsymbol{d}(\boldsymbol{m}^T\boldsymbol{e}_1)^2 - \boldsymbol{e}_1^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{e}_2\boldsymbol{m}^T\boldsymbol{e}_1 - MN\boldsymbol{e}_2^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{m} + MN\boldsymbol{m}^T\boldsymbol{d}\boldsymbol{m}^T\boldsymbol{e}_2 \end{bmatrix} , \tag{2.29}$$

where

$$\det \boldsymbol{M} = MN\left(MN\boldsymbol{m}^T\boldsymbol{m} - (\boldsymbol{m}^T\boldsymbol{e}_1)^2 - (\boldsymbol{m}^T\boldsymbol{e}_2)^2\right) . \tag{2.30}$$

We may re-write these as (2.19)–(2.21). $\square$

3 Method

This chapter explains the process and methodology used in our research, presented in a largely chronological order with thematic groupings. It provides an account of the steps taken in the process.

3.1 Phase One: Exploration and Preparation

We started our research process by collecting data and preparing the measurement setup.

3.1.1 Literature Study

In the introductory section of our thesis we started with a comprehensive literature review. This phase provided us with a deeper understanding of key concepts such as homographies and pose prediction, which are essential for our research. Our research involved building on a a server-client setup where the server, based on WebRTC, processed video feeds and pose data, and the client, built in Unity, visualized the data in real-time. Our predictor used homography calculations to adjust the positions of virtual elements based on head movements, reducing perceived latency. Some of our findings are presented in related work section, including a notable study on homography-based loss functions for camera pose regression. This study demonstrates how homography can be effectively utilized for accurate pose estimation in augmented reality applications [4].

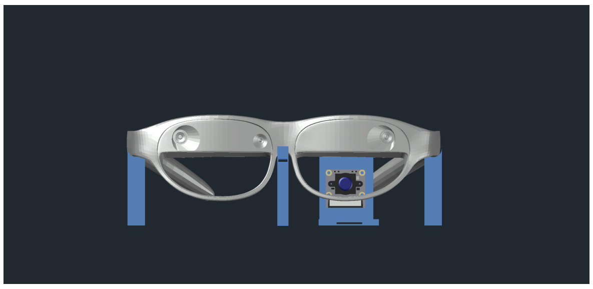


Figure 3.1.1 CAD-model of the Nreal Air holder.

3.1.2 Hardware Setup and Test Environment

In this study, we utilized two different types of head-mounted displays (HMDs) to examine and validate our prediction algorithm. The setup included:

- Nreal Air glasses [32]: These are optical see-through augmented reality (AR) HMDs that feature a transparent display. This setup allows users to see the real world while digital content is superimposed on their view.
- Varjo XR3 [29]: These are primarily virtual reality (VR) HMDs but also include video see-through capabilities, making them suitable for AR applications.

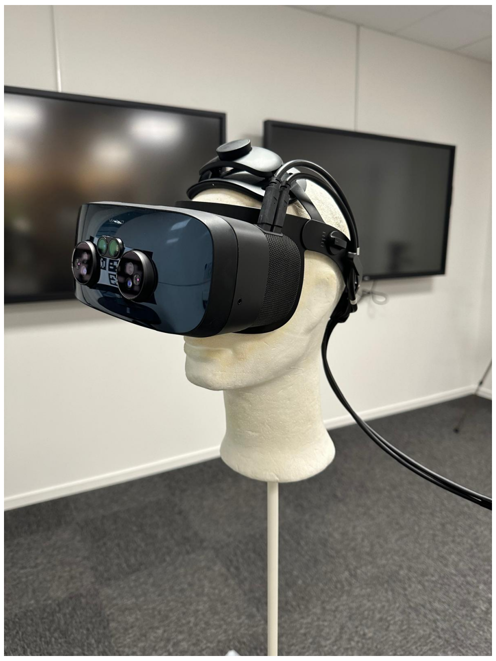
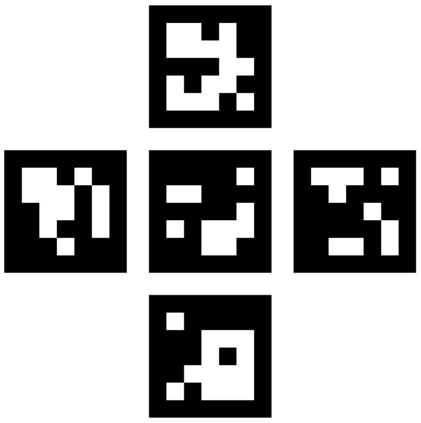


Figure 3.1.2 Varjo XR3 holder setup.

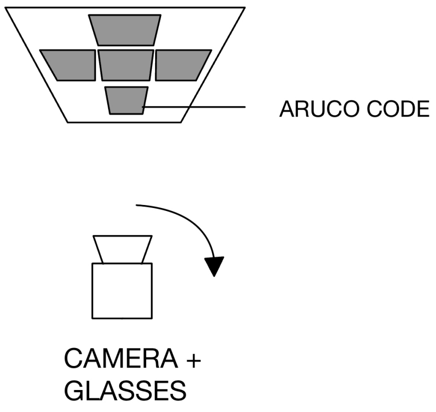
Additionally, we designed custom mounts for these devices:

- Mount for Nreal Air glasses: We developed a 3D printed holder for the Nreal glasses to facilitate stable simulations of head movements. This mount was placed on a tripod to ensure a consistent and reproducible setup. Behind the glasses, a Raspberry Pi camera was positioned to capture the testing process, providing detailed documentation and analysis of our algorithm's performance under various conditions. The 3D model can be found in Figure 3.1.1.

- Mount for Varjo XR3: To evaluate our prediction algorithm using the Varjo headset, we used a styrofoam head model seen on Figure 3.1.2. This model, borrowed from the VR lab at Lund University, helped us create a testing environment that is stable enough for our needs. The setup is complete with two Valve Index Base Station lighthouse trackers [27], that the device uses for spatial positioning alongside its IMU.



(a) Aruco code setup used.



(b) Setup for latency measurement.

Figure 3.1.3 Set up for the experiment.

3.1.2.1 Raspberry Pi Setup for Optical See-Through

We equipped a Raspberry Pi 4 Model B with its camera module, positioned as shown in Figure 3.1.1, peering through one of the lenses. We installed Raspberry Pi OS on the microcontroller, and a Python script was utilized for recording videos and displaying them through the Pi's HDMI connection.

3.1.3 Creating a Latency Simulator

For an initial demonstration, regarding the goal of this project, we created a latency simulator. As seen in Figure 3.1.4, we introduced the concept of a red cross representing a real-world point, and a blue cross representing a corresponding offloaded point. The demo shows the blue cross following the red cross with a customizable delay, viewable through the Nreal Air AR glasses. This serves to illustrate the initial problem with offloading in AR, as well as for us as developers to have a starting point for the project.

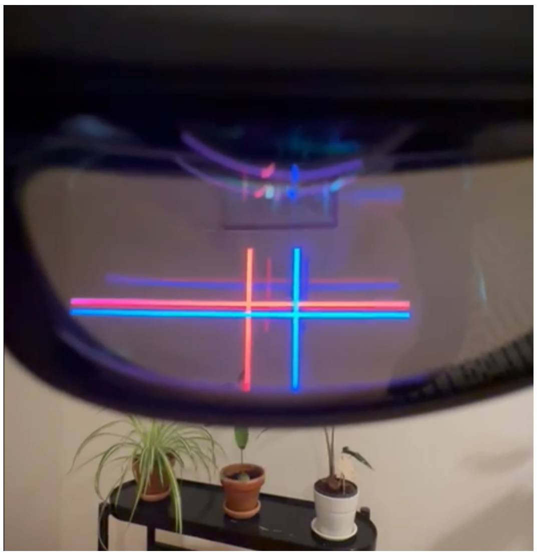


Figure 3.1.4 Latency simulator through the lens of Nreal Air.

3.1.4 Measuring Baseline Noise and Latency

To ensure the accuracy of the position and rotation data being sent from the HMDs was crucial. To verify this, we conducted a test where we measured the noise affecting these parameters when the headsets remained static. Although we initially included the Nreal Air in our tests, it malfunctioned and was not used in the final analysis. Therefore, we are only presenting the results for the Varjo headset here.

For this test the headset remained on its holder undisturbed for approximately a minute. In Unity a custom script created a text file containing the pose data (namely: translation and rotation). Later we visualized this data using a Python script.

The results can be seen in Figures 3.1.5 and 3.1.6. The standard deviation, as shown in Table 3.1, is within acceptable boundaries (one Unity unit corresponds to one meter in physical space [26]).

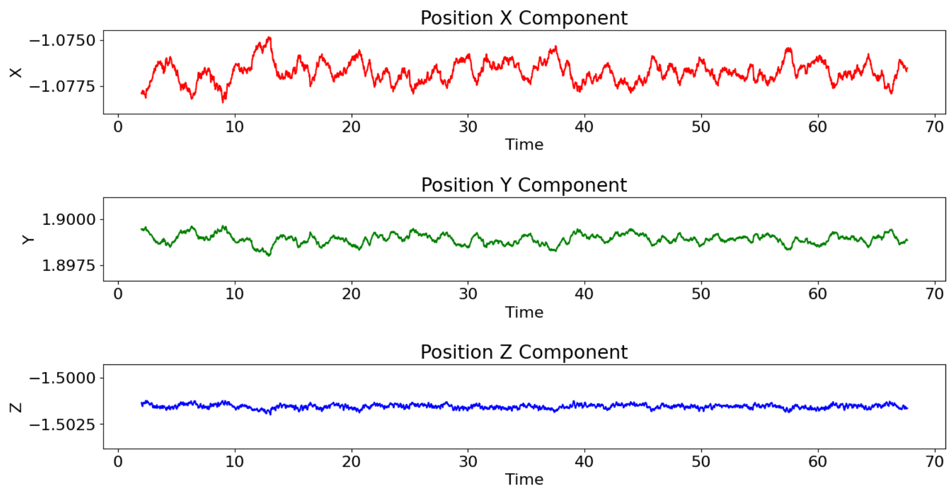


Figure 3.1.5 Baseline noise affecting head position for Varjo XR3.

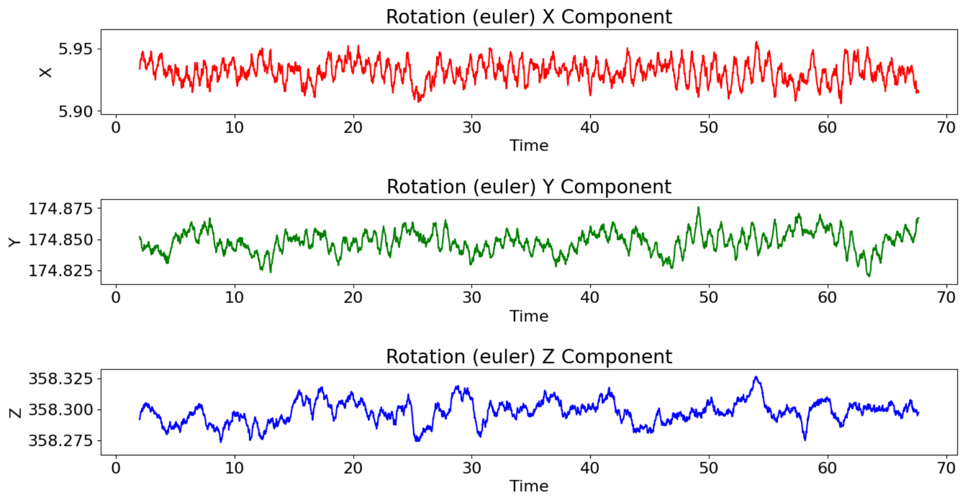


Figure 3.1.6 Baseline noise affecting head rotation for Varjo XR3 (degrees).

Table 3.1 Standard Deviation of Positional and Rotational Components (Varjo XR Noise Level).

Component	Standard Deviation
Position X (Unity units)	$5.7 \cdot 10^{-4}$
Position Y (Unity units)	$2.7 \cdot 10^{-4}$
Position Z (Unity units)	$1.1 \cdot 10^{-4}$
Rotation X-axis (degrees)	$8.53 \cdot 10^{-3}$
Rotation Y-axis (degrees)	$8.67 \cdot 10^{-3}$
Rotation Z-axis (degrees)	$9.01 \cdot 10^{-3}$

3.2 Phase Two: Implementation

After completing the preparations, we began developing the necessary functionalities for our research.

3.2.1 Developing the Server

The server we used for interfacing with the headset is based on WebRTC. This server program is written in Python, using Python's aiortc library, which implements webRTC functionality [1].

WebRTC (Web Real-Time Communication) is an open-source project that enables web browsers and mobile applications to communicate in real-time via simple APIs without requiring an intermediary or additional plugins [30]. It is widely used from applications such as video calls to computer games.

The server receives a video feed from the XR headset and can optionally handle metadata, a feature we utilized in our setup. It returns metadata that can be used on the headset to visualize necessary information for specific applications, such as the locations of bounding box corners for a semantic segmentation algorithm.

A critical component of our setup is the integration of Aruco marker detection. Aruco markers are black and white markers (seen in Figure 3.1.3a (b)) that can be easily detected and tracked in real-time by a camera, making them highly useful for AR applications where precise positioning and orientation are needed [33].

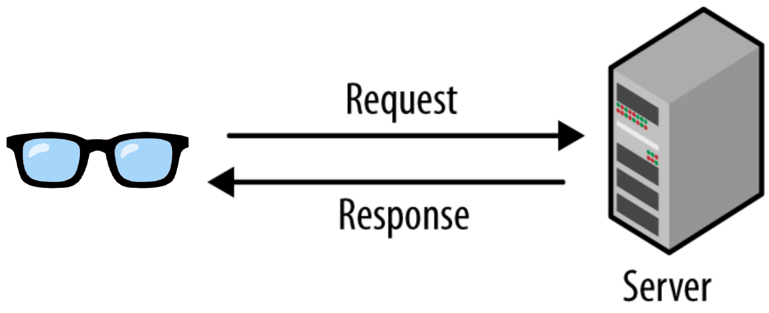


Figure 3.2.1 Server-Client Communication [12].

In our implementation, the camera continuously captures the video feeds and sends them to the server for detection. Once the Aruco codes are detected, the server processes this data to determine the positions and orientations of the markers in pixel coordinates. After processing, the server compiles this spatial data into metadata which is then sent back to the client. This metadata includes the coordinates of the midpoint of each Aruco marker and their respective unique identifiers. The detection of the Aruco markers is performed in real-time, ensuring continuous and accurate tracking, which is then relayed back to the client.

To simulate real-world network conditions, we utilize a network condition simulator called "clumsy". Clumsy is a software tool that allows us to introduce various types of network issues like latency, packet loss, or bandwidth restrictions [19]. By incorporating clumsy into our test setup, we can artificially introduce specific amounts of latency, thereby mimicking less-than-ideal network conditions that users might experience in actual application scenarios.

3.2.2 Developing the Client

For the client side we also built on a custom application, meant to interface with the Python server. This application is developed using Unity and comes in various versions tailored to different headsets. For instance, the Nreal Air 2 uses an Android build, while the Varjo XR3 operates on a Windows application version, or directly from Unity.

The application transmits a video feed, with optional metadata, and receives metadata from a Python server, which it then visualizes in XR. Due to a malfunction with the Nreal Air AR glasses during our research, we exclusively used the Varjo version for evaluation. Although initially we ran the prediction algorithm on the server side due to its support for OpenCV, we eventually shifted the prediction to the client side, as this is the logical placement for effectively combating latency. This transition is further detailed in Section 3.2.4.

3.2.3 Creating a Predictor

To predict the rendering location of certain pixels, we employed a homography-based estimation method. This approach was selected because our scene projection involves mapping three-dimensional points onto a two-dimensional plane. By using homographies, we can effectively relate transformations between two planes corresponding to different camera views or poses. This process utilizes the user's head movement, among other factors, to calculate a homography. This calculation is then used to adjust the points received from the server (which are affected by latency) to positions that more closely align with the real-world objects referenced by the AR application. In this thesis, we refer to this process as prediction because it provides approximate information about data we do not yet have access to. However, it is important to note that this prediction does not forecast future events, but rather infers current unknowns.

One way to grasp this concept is to consider that, in the field of computer vision, this method is typically applied in reverse [14]. Traditionally, by analyzing the positions of pixels across two images, we can determine the position of the camera that captured these images. In our case, we employ this method in reverse, i.e, (3.1).

$$\boldsymbol{p}_i' \sim H\boldsymbol{p_i} \quad \Leftrightarrow \quad \boldsymbol{p}_i \sim \boldsymbol{H}^{-1}\boldsymbol{p}_i \, . \tag{3.1}$$

For the homography calculation, as seen in (2.10), we also need an $\boldsymbol{n}$ and d, which describe the normal of the plane (on which the points lie) and the distance from the plane to the camera frame, respectively. These can be calculated in Unity, where this plane is an invisible game object. For the purposes of this thesis we kept this plane in a location that is aligned with Unity's coordinate system, where $\boldsymbol{n} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T$. The d variable is the camera's distance to this plane.

Python's robust OpenCV support, which we had no access to in Unity, led us to initially develop the prediction algorithm on the server. We streamed a video to the server, where we first recognized the five Aruco codes and then captured their central coordinates for our calculations. Along with the video stream, we transmitted pose data from Unity, including the camera position and rotation of the Nreal Air glasses' center (streaming) camera within Unity's coordinate system.

Given that Unity and OpenCV utilize different coordinate systems, where OpenCV's y-axis is flipped, a correction is necessary (shown in Figure 3.2.2). After making this adjustment, we calculated the homography. Using this homography, we transformed the points. We initially grabbed the Aruco coordinates and applied the homography to transform them. By comparing these transformed coordinates to the original Aruco coordinates, we could confirm the effectiveness of the prediction, as the points tended to align well.

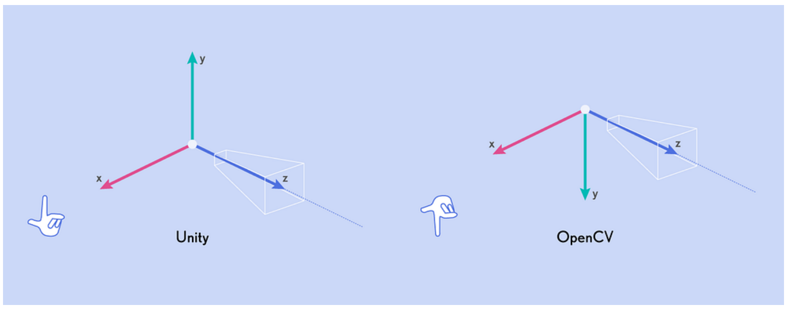


Figure 3.2.2 Unity's coordinate system (on the left) and OpenCV's coordinate system (on the right) [20].

Our steps for this initial version summarized:

1. Stream video to the server, recognize Aruco codes to capture their central coordinates.

2. Send pose data from Unity, containing the Nreal Air glasses' camera position and rotation, in Unity's coordinate system.

3. Correct for the differences in the coordinate systems used by Unity and OpenCV.

4. Calculate the homography based on the corrected data.

5. Use the homography to transform the initial Aruco center coordinates.

6. Compare the transformed coordinates with the original Aruco coordinates to validate the accuracy of the prediction.

3.2.4 Running the Prediction On-Device

After initially developing our predictor on the server, the next step involved implementing the same process in Unity and using it on-device. For this transition, we replicated the steps of the server implementation. However, in Unity, we had to manually write several functions that were readily available in the Python implementation.

Firstly we successfully developed the same functionality on the client side, namely: being able to compare the predicted points to the ground truth Aruco codes. We shall refer to this version as version 1. This version works for describing static points in the real world. However, for objects that move over time this approach would not be feasible. We evaluate the versions further in Section 3.3.

For version 2, we are updating the position of the predicted points whenever we are receiving new metadata from the server. Visually this means that the blue cross is "jumping" to overlap with the red cross whenever the new Aruco code positions are received, and when no metadata arrives it is behaving similarly to version one. This in theory would allow this version to describe non-static points in space, however in practice this is just an intermittent step between version 1 and 3 that is not quite useful in its application. The reason for this is that the approach does not account for the possibility that by the time the metadata is received by the client, the user may have changed their orientation. As a result, the new metadata could correspond to an inaccurate position, which the predicted points should not be moved to. Also, aligning the predicted points with this metadata can cause the blue cross to "jump", which causes a subpar user experience.

We aimed to complete our implementation with version 3, developing the functionalities missing from version 2. First, we aimed to fix the problem with the received metadata (red cross coordinates) being affected by latency. In order to find where these points should be rendered in reality, we have to know the pose when the frame was taken and the pose when it arrives. With this we can use our prediction algorithm to find the correct position based on how much the user's head has moved between these two points in time.

For this purpose, the user's historical head pose data was required, and, to this end, we implemented a dictionary to store the necessary information. This is when we ran into difficulties with Unity's WebRTC implementation. For a perfect solution we needed to attach metadata to the frames, which would have stored the user's pose. This way the poses would be synchronized with their respective images. However, we found no way to achieve this with the current state of the WebRTC library as of the writing of this paper. Instead, we had to resort to trying to measure this latency using the latency of a separate metadata channel, however since these are sent asynchronously, these values are most likely not the same. We can then use this metadata latency to estimate the latency of the video stream.

In order to measure the latency of the metadata channel, we created a dictionary storing the IDs of messages with the timestamp of when they are being sent. Then when we receive metadata we use its embedded ID to look up the time it was sent, and calculate latency using the current time. We are able to apply a multiplier to this latency to get a rough estimate of the one affecting the video stream. Then we have an estimate of when the image, the one we are receiving metadata about, was taken.

Having a rough timestamp of when the image was taken, we can use this to find the closest timestamp stored in our time-pose dictionary. Thus we get an approximation of the original pose, and we use use our prediction algorithm on this data. The result is an approximation of where the real-world points are now, looking through the HMD, assuming that the aforementioned points haven't moved since the image was sent. Due to this and the inaccuracies previously described, the result of this approach still causes slight "jumps" where the predicted points suddenly shift. This can detract from the user experience and become visually distracting when the frequency of incoming data is low. To combat this we applied LERP between the current position of the predicted point and its "new" position (the metadata received at that moment). Where the final result should lie between these two points can be fine-tuned with a variable in the application.

Finally, we developed a (to our knowledge) novel approach for calibrating between virtual reality and virtual space, presented in Section 2.8. With this calibration our prediction algorithm's accuracy improved further.

In summary:

- **Version 1**: Only feed the prediction algorithm the point coordinates from the first frame. After that run prediction based on this and head movement, and compare these to the ground truth. (*The blue cross is based solely on head movement after the first frame)*
- **Version 2**: Whenever data is received, move the predicted points to that position. In the frames where data is not received is when the prediction algorithm is running to "fill the gaps". (*The blue cross "jumps" to the red cross when new metadata arrives from the server. When the red cross is not updated the blue cross updates based on the prediction algorithm*).
- **Version 3**: When data is received from the server, move the predicted points not directly to the received coordinates, but to the position those coordinates were at the time the frame was recorded (with imperfect accuracy because of technicalities). Also applying LERP to increase viewing comfort, and virtual-to-real-world calibration for increased accuracy. (*The blue cross moves in a similar way as with version 2, however here the blue cross doesn't "jump" to the exact position of the red cross, instead it moves toward the direction of the real-world point that the red cross represents.*)

3.2.5 Developing for the Varjo XR3

In order to get an accurate measurement wireless connections were not ideal, as the latency can fluctuate considerably (how much exactly we could not measure because of the malfunction detailed below). To combat this, we started implementing our project on the wired Varjo XR3 headset, with the plan of injecting latency artificially to this wired connections. For this, we simply ported the Nreal project and re-implemented the Nreal-specific parts for the Varjo headset. This process was mostly straightforward, except for accessing the camera feed, which is not natively supported for the Varjo headset and required a custom script. During the later stages of our process, the Nreal glasses stopped being recognized by the device despite confirming the hardware was functional. Consequently, we were unable to use them, and had to proceed exclusively with the Varjo XR3 for evaluating our algorithm., thus the Varjo XR3 is the only device we were able to continue with and evaluate our algorithm on.

Earlier in the thesis process we modeled and 3D-printed a holder for the Nreal Air glasses, seen in Figure 3.2.3. Because of the glasses' malfunction this was never used for testing. Later we assembled a simple testing environment for the Varjo HMD, seen in Figure 3.2.4.

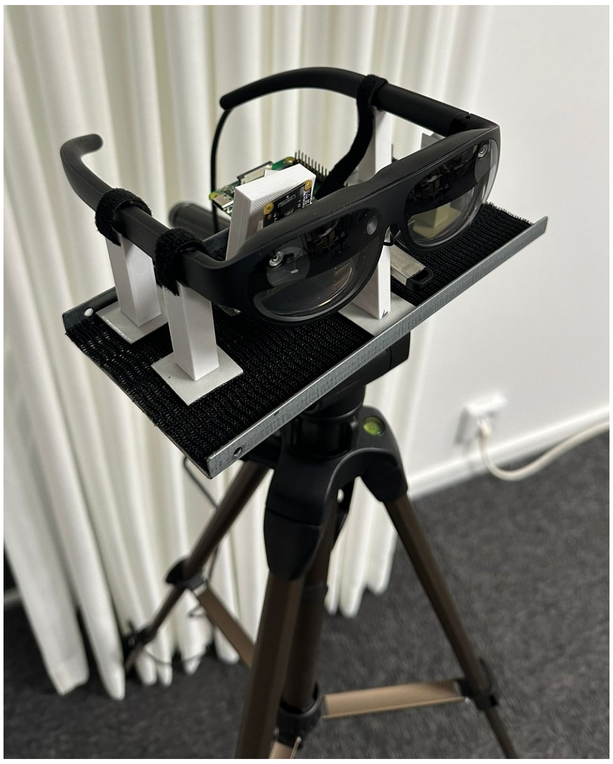


Figure 3.2.3 Nreal holder.

3.3 Phase Three: Evaluation

Throughout the development process, we repeatedly evaluated our algorithm, making this a phase we revisited multiple times. For version one it was crucial to see that the homography calculated from the poses $\boldsymbol{H}_{\text{pose}}$ is correct, which we examined by comparing it to the homography calculated from image points. After we deemed these reasonably close, we visually inspected the points looking for the behaviour we expect, which we found: the points were consistently aligning with the ground truth markers. The match was not perfect, which could be due to noise, imperfections in our approach, or other unknown sources.

We evaluated version two in a similar manner after completing our transition to Unity on the client side. Once we saw that the predicted points on the server and the client aligned we knew that the transition was successful.

In-depth evaluation only occurred with the finished algorithm, version 3. These are the results discussed in Chapter 4. To evaluate the accuracy of our algorithm, we analyzed 20-30 pictures taken over a span of 2-3 seconds, simulating a short period

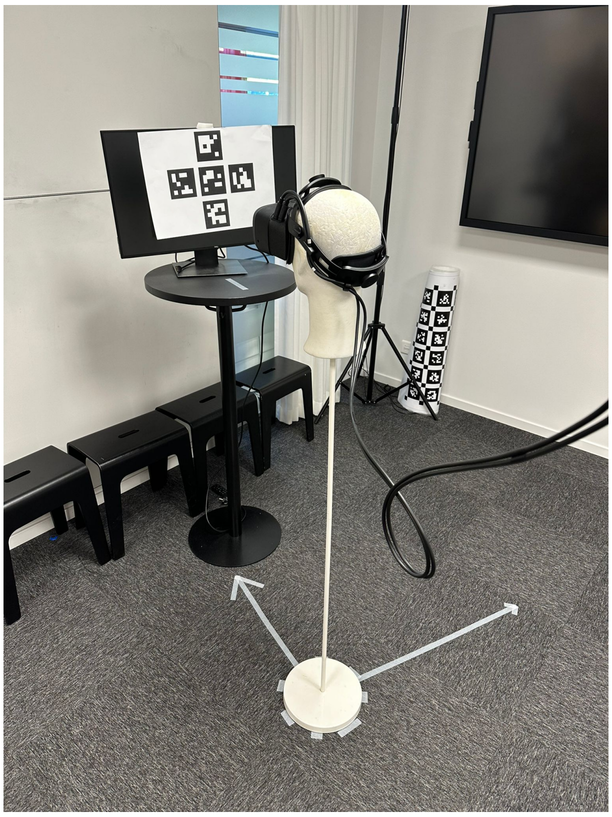
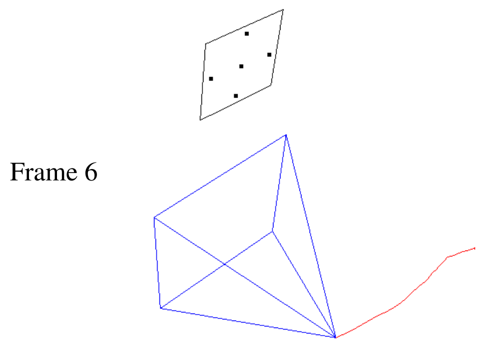


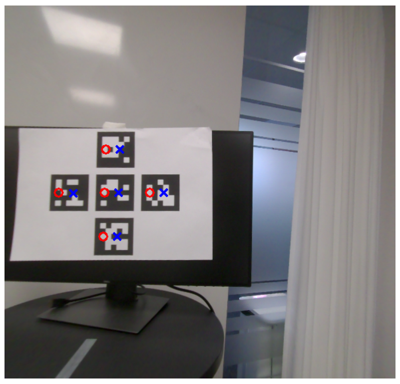
Figure 3.2.4 Hardware setup.

of head movement. We conducted this process by aiming for "pure" translation and "pure" rotation movements. Although achieving these ideal movements perfectly is not feasible with our setup, these attempts provide an approximation suitable for our needs.

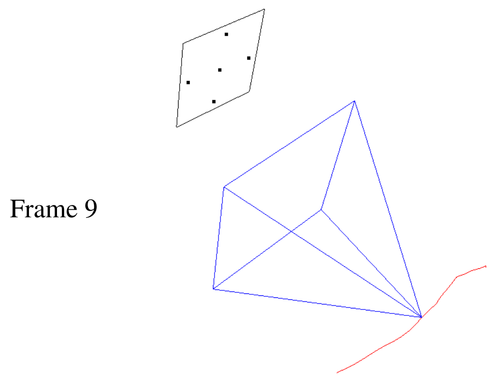
The Varjo headset was placed on a holder 0.75 meters away from a paper with the Aruco code cross. For the testing itself we either rotated the "head" horizontally, or translated the entirety of the holder along the *x* (horizontal) axis, all while recording frames and poses. Some example images are visible in Figure 3.3.1 and Figure 3.3.2: these figures show the results of our prediction algorithm for varying distances, with (b) representing a short distance and (f) representing a long distance. The corresponding camera trajectories can also be seen next to their respective result images. The frame numbers are indicated, assuming a frame rate of 60 frames per second (fps), with the final image showing around 100-200 ms latency.



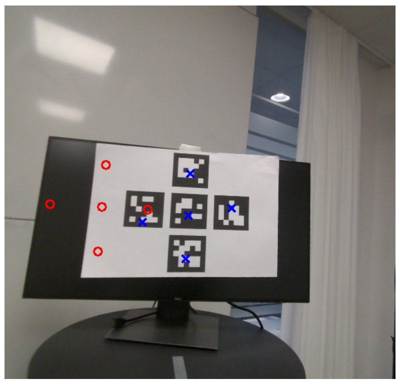
(a)



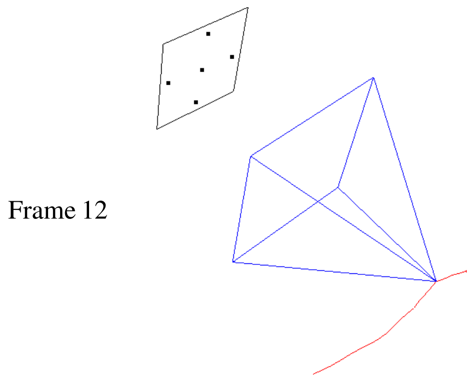
(b)



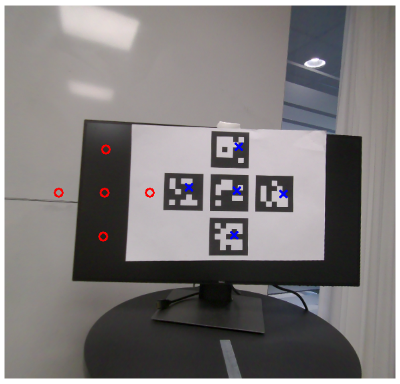
(c)



(d)



(e)



(f)

Figure 3.3.1 Three examples of the algorithm result with camera trajectories—translation.

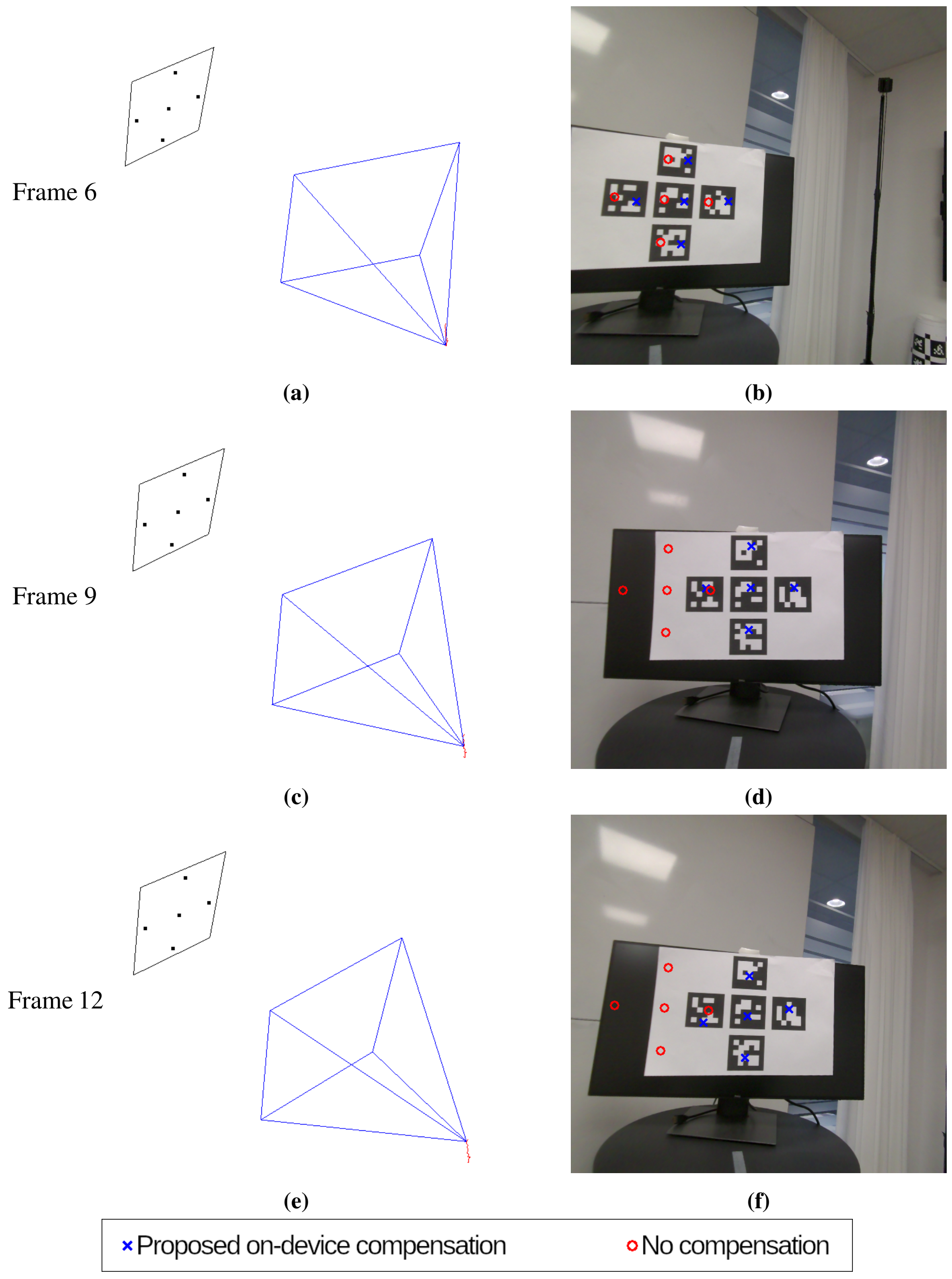


Figure 3.3.2 Three examples of the algorithm result with camera trajectories—rotation.

4 Results

This chapter presents the findings of our research.

Based on the setup and process presented in Section 3.3, Figure 4.0.1 and Figure 4.0.2 show the distance in real-world and virtual point coordinates with and without our prediction algorithm. This is based on the same data as in Figure 3.3.1 and Figure 3.3.2, where we sample one frame per 0.1 seconds. We can see that our approach significantly decreases the distance between the real-world and virtual points.

One way to think about these results is: if the frequency of incoming metadata is very low or the user makes quick head movements, the prediction can resemble what is shown in Figure 3.3.1(image f), which demonstrates a significant improvement compared to not using compensation. Conversely, if the frequency is high or the movements are slow, the compensated points remain close to the uncompensated ones, meaning the prediction will not enhance the experience as dramatically.

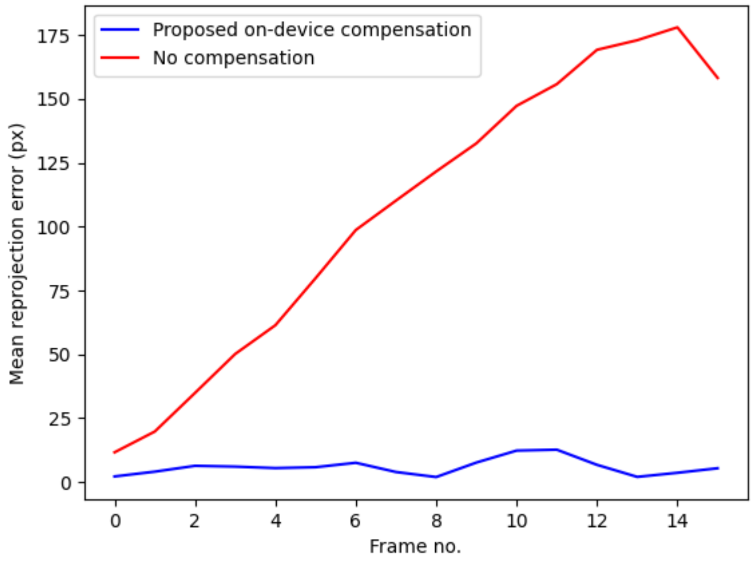


Figure 4.0.1 Mean reprojection error with no compensation (red graph) compared to with compensation (blue graph): Translation.

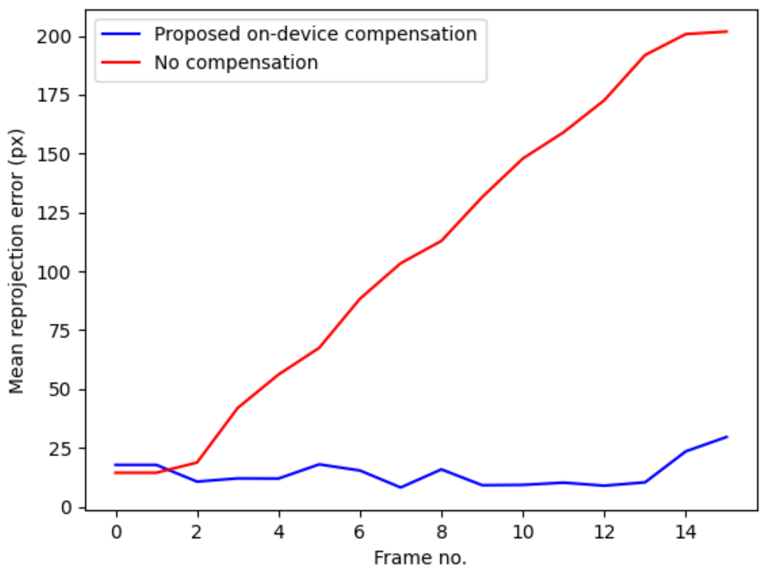


Figure 4.0.2 Mean reprojection error with no compensation (red graph) compared to with compensation (blue graph): Rotation.

5 Discussion

In this chapter, we discussed an in-depth analysis and interpretation of our research findings, alongside addressing limitations and outlining future directions.

5.1 Discussion of Results

Our research demonstrated the feasibility and effectiveness of using homography-based prediction through pose information to mitigate latency in augmented reality (AR) applications. The predicted points aligned well with the ground truth markers, validating the accuracy of our algorithm.

The key findings are:

- **Accuracy of Predictions**: The predicted points showed a significant improvement over the "red-cross" positions with no prediction applied, particularly during periods of fast head movement or low metadata frequency.
- **Implementation Challenges**: The transition of the prediction algorithm from the server to the client-side (Unity) was successful, but highlighted challenges with synchronizing metadata and video frames, particularly due to limitations in Unity's WebRTC implementation.

Figures 3.3.1 and 3.3.2 illustrate the results of various scenarios of predicting points during head movements, and Figures 4.0.1 and 4.0.2 show numeric qualities of our approach.

5.2 Evaluating Our Process

The development and evaluation process revealed several insights:

- **Initial Server Implementation**: Implementing the algorithm on the server first allowed rapid development and testing using Python's robust OpenCV support. This step was crucial for validating the homography calculation before transitioning to the client-side.

- **Transition to Unity**: Porting the algorithm to Unity presented challenges, particularly with accessing the camera feed on the Varjo XR3 and aligning the coordinate system differences between Unity and OpenCV.

- **Testing and Evaluation**: The structured approach to testing, including baseline noise and latency measurements, ensured that our results were reliable. Using a controlled environment and consistent setups was essential for accurate measurements.

- **Lack of User Testing**: We did not conduct user testing, as our focus was primarily on the theoretical aspects and developing a robust algorithm. We will discuss this further in Section 5.3.

While our methodology was effective, there were areas where different approaches could have been beneficial. For instance, better synchronization of metadata and video frames in Unity could have been achieved with alternative communication protocols.

5.3 Limitations

Our study encountered several limitations that impacted its comprehensiveness. Our major setback occurred during the testing with hardware phase due to the malfunction of the Nreal glasses. Despite our initial plan to utilize both the Nreal glasses and the Varjo XR series for comparative analysis, the unexpected failure of the Nreal glasses disrupted our ability to conduct comprehensive testing. Consequently, we were unable to evaluate our prediction algorithm's performance on the Nreal platform. We chose these two platforms specifically to explore the differences between video see-through and pass-through technologies, aiming to provide a comprehensive analysis of latency compensation methods across different XR platforms. Additionally, Varjo's superior processing power significantly reduces rendering time. Consequently, future pose prediction could have had a more substantial impact on Nreal glasses, as their rendering time is considerably longer.

Another limitation of our study is the lack of user testing. While we do not believe this affects the overall quality of the information presented, a user study would be informative, particularly as we are discussing perceived latency. The user test could provide information about latency's effect on user experience when it is uncompensated, compared to when our compensation is applied, for example. Some user test ideas are presented in Section 5.5.

5.4 Addressing Research Questions

Our research aimed to address the following questions:

1. How can an algorithm reduce latency in offloaded AR/MR applications by compensating for pose differences using sensor data?

We developed a homography-based prediction algorithm that uses sensor data from head-mounted displays to compensate for pose differences and reduce latency. The algorithm predicts the position of real-world points in the virtual environment, reducing the perceived latency in offloaded AR/MR applications.

- **For what types of motions and applications is this algorithm effective?** The algorithm is effective for a variety of head movements, including translations and rotations. The effectiveness of our approach was demonstrated through tests involving both pure translation and pure rotation.

2. How can calibration between the physical and virtual worlds be achieved for accurate pose-based prediction? To integrate physical and virtual spaces for accurate pose-based prediction, we needed to ensure that our pose data, which was in Unity space, aligned with our physical setup. Here's how we approached this calibration:

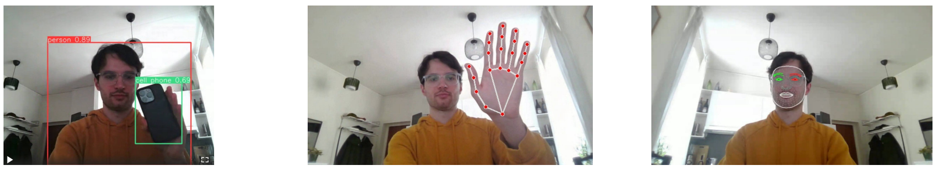
- **Marker Detection** We began by detecting Aruco markers in the captured images using a predefined dictionary and detection parameters. The detected marker corners provided the basis for subsequent calculations.
- **3D Points in Unity** In the Unity environment, we placed virtual 3D cubes at the same locations as the physical Aruco markers. These cubes provided known 3D points anchored to the Unity coordinate system.
- **Pose Data From Unity** For each captured image, we retrieved the corresponding pose data from Unity, which included the position and rotation in the form of quaternions. This pose data was used to project the 3D anchored points into the image plane.
- **Midpoint Calculation and Image Adjustment** We calculated the midpoints of the detected Aruco markers. Due to differences in coordinate systems between Unity and the image plane, we flipped the image horizontally and vertically to align the midpoints correctly.
- **Camera Calibration $\boldsymbol{K}$** To ensure accurate alignment, we estimated the intrinsic camera calibration matrix, denoted as $\boldsymbol{K}$. This matrix was optimized

to transform coordinates from the physical space to Unity space accurately. The matrix $\boldsymbol{K}$ was calculated by solving a system of linear equations derived from the correspondences and pose data, ensuring minimal error. The details of this derivation and optimization process are discussed thoroughly in the theory section.

It is also important to note that our approach can provide even more accurate results for applications where tracking real-world objects is not required, instead entirely virtual objects are being visualized. As long as we can keep our assumption of the object's movement aligning to a single plane, these use cases would be more accurate because we can eliminate the inaccuracies stemming from the noise of aligning the virtual and real worlds.

By following these steps, we achieved a robust calibration that accurately aligned the physical and virtual worlds, allowing for precise pose-based prediction in the Unity environment. To the best of our knowledge, this approach is novel and has not been done before.

With this calibration matrix $\boldsymbol{K}$, we can now easily perform homography-based pose estimation. The matrix $\boldsymbol{K}$ is essential for reliable pose prediction because it accurately captures the intrinsic parameters of the camera, such as focal length and principal point. This allows us to transform coordinates between physical and virtual spaces with high precision, ensuring that the projected 3D points and detected marker positions align perfectly. As a result, our pose predictions become highly accurate and reliable, enhancing the overall effectiveness of our virtual-physical integration.



(a) Semantic Segmentation **(b) Hand detection** **(c) Face mesh**

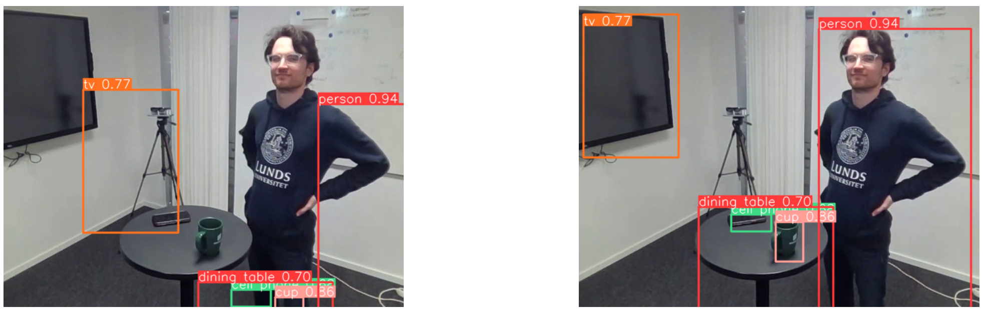
Figure 5.4.1 Augmented reality applications.

5.5 Future Work

In the future, we will apply our prediction algorithm to existing AR applications, including semantic segmentation, face mesh, and hand tracking (as seen in Figure 5.4.1). By integrating our prediction algorithm into these applications, we will aim to reduce the latency due to offloading these tasks which would lead to enhancing the user experience. Conducting user studies will be the most important aspect of this future work. We are interested in understanding how users perceive the impact of our algorithm on latency reduction. Specifically, we will investigate whether users find that our algorithm effectively reduces latency or if they perceive it as overshooting, potentially causing disorientation or other negative effects.

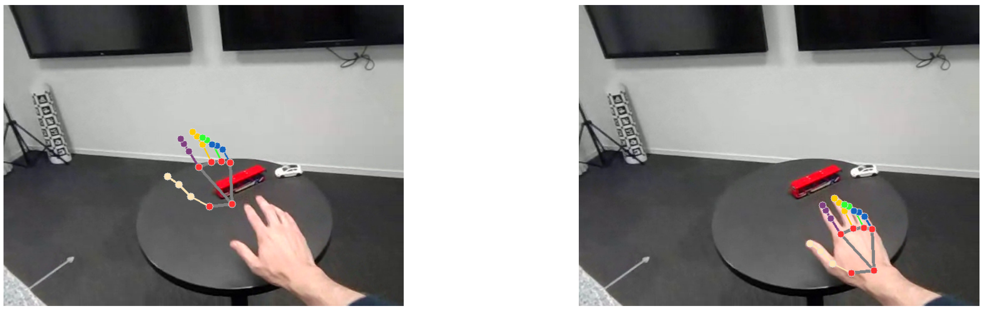
Furthermore, we are interested in exploring the sensitivity of different applications to latency. Through user testing and feedback, we will identify which applications are more sensitive to latency and to what extent. For example, we expect that applications such as face mesh would be more sensitive to compared to semantic segmentation, as we are more sensitive to slight changes around a person's face than to those of of a bounding box. Understanding the threshold at which latency becomes noticeable or intolerable for each application will be valuable in optimizing our predictive algorithm and informing future AR application development.

Based on our findings detailed in Chapter 4, some preliminary conclusions can be drawn about how our approach could benefit the augmented reality applications discussed in this chapter. Assuming a frame rate of 30 frames per second and noting that our sample data for rotation and translation spans approximately 15 frames, we estimate a latency of around 500 ms. This can be considered a scenario with poor network conditions, which would significantly diminish the user experience. Assuming significant head movement in this time frame, we can expect similar offsets compared to our findings. We use the translation data as a reference and apply these offsets to the overlays. The results can be seen in Figures 5.5.1 to 5.5.3. The user experience of these applications equipped with the algorithm detailed in this paper will be further examined in a user study.



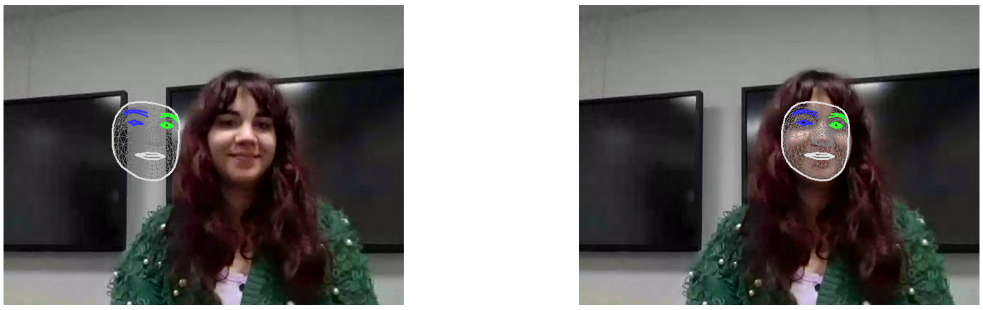
(a) Without prediction **(b) With prediction**

Figure 5.5.1 Compensation for estimated latency in semantic segmentation.



(a) Without prediction **(b) With prediction**

Figure 5.5.2 Compensation for estimated latency in hand detection.



(a) Without prediction **(b) With prediction**

Figure 5.5.3 Compensation for estimated latency in face mesh.

6 Conclusion

Summarizing our process and findings.

In this thesis, we explored the development and implementation of a homography-based prediction algorithm to reduce latency in offloaded augmented reality (AR) applications. Here are the main takeaways from our research:

- **Effective Latency Reduction:** Our homography-based prediction algorithm effectively reduces the perceived latency in AR applications by compensating for pose differences. The algorithm showed significant improvements in aligning virtual and real-world objects.
- **Robustness Across Movements:** The algorithm is effective for various types of head movements, including translations and rotations. This versatility makes it suitable for a wide range of applications, such as navigation, remote assistance, and interactive gaming.
- **Successful Calibration:** We successfully achieved calibration between the physical and virtual worlds using Aruco marker detection and homography calculation. This calibration ensures accurate pose-based predictions, enhancing the precision and reliability of AR experiences.

In conclusion, our research provides a comprehensive framework for reducing latency in offloaded AR applications through homography-based pose prediction. Future work can build on these findings to further enhance the algorithm's accuracy and extend its applicability to more dynamic environments and diverse use cases.

References

[1] aiortc developers. *Examples - aiortc*. Accessed: 2024-05-17. URL: https://aiortc.readthedocs.io/en/latest/examples.html.

[2] R. T. Azuma. "A survey of augmented reality". *Presence: teleoperators & virtual environments* **6**:4 (1997), pp. 355–385.

[3] R. T. Azuma. *Predictive tracking for augmented reality*. The University of North Carolina at Chapel Hill, 1995.

[4] C. Boittiaux, R. Marxer, C. Dune, A. Arnaubec, and V. Hugel. "Homography-based loss function for camera pose regression". *IEEE Robotics and Automation Letters* **7**:3 (2022), pp. 6242–6249.

[5] K. Boos, D. Chu, and E. Cuervo. "Flashback: immersive virtual reality on mobile devices via rendering memoization". *GetMobile: Mobile Computing and Communications* **20** (Apr. 2017), pp. 23–27. DOI: 10.1145/3081016.3081026.

[6] D. Brown. "Decentering distortion of lenses". *Photogrammetric engineering* **32**:3 (1996), pp. 444–462.

[7] K. Cao, Y. Liu, G. Meng, and Q. Sun. "An overview on edge computing research". *IEEE Access* **8** (2020), pp. 85714–85728. DOI: 10.1109/ACCESS.2020.2991734.

[8] E. Chang, H. T. Kim, and B. Yoo. "Virtual reality sickness: a review of causes and measurements". *International Journal of Human–Computer Interaction* **36**:17 (2020), pp. 1658–1682.

[9] D. Cheng, Q. Wang, Y. Liu, H. Chen, D. Ni, X. Wang, C. Yao, Q. Hou, W. Hou, G. Luo, et al. "Design and manufacture ar head-mounted displays: a review and outlook". *Light: Advanced Manufacturing* **2**:3 (2021), pp. 350–369.

[10] H. Collewijn and J. B. Smeets. "Early components of the human vestibulo-ocular response to head rotation: latency and gain". *Journal of Neurophysiology* **84**:1 (2000), pp. 376–389.

[11] E. Cuervo, A. Wolman, L. P. Cox, K. Lebeck, A. Razeen, M. Musuvathi, and S. Saroiu. "Kahawai: high-quality mobile gaming using gpu offload". In: *MobiSys'15*. ACM – Association for Computing Machinery, May 2015. URL: https://www.microsoft.com/en-us/research/publication/kahawai-high-quality-mobile-gaming-using-gpu-offload/.

[12] D. Darab. *RESTful API*. Accessed: 2024-05-17. 2020. URL: https://darvishdarab.github.io/cs421_f20/docs/readings/restful/api/.

[13] J. Fu, A. Rota, S. Li, J. Zhao, Q. Liu, E. Iovene, G. Ferrigno, and E. De Momi. "Recent advancements in augmented reality for robotic applications: a survey". In: *Actuators*. Vol. 12. 8. MDPI. 2023, p. 323.

[14] R. Hartley and A. Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[15] I. Howard. "Human visual orientation(book)". *Chichester, Sussex, England and New York, John Wiley and Sons, 1982. 704 p* (1982).

[16] L. Hsiao, B. Krajancich, P. Levis, G. Wetzstein, and K. Winstein. "Towards retina-quality vr video streaming: 15ms could save you 80% of your bandwidth". *ACM SIGCOMM Computer Communication Review* **52**:1 (2022), pp. 10–19.

[17] G. K. Illahi, A. Vaishnav, T. Kämäräinen, M. Siekkinen, and M. Di Francesco. "Learning to predict head pose in remotely-rendered virtual reality". In: *Proceedings of the 14th Conference on ACM Multimedia Systems*. 2023, pp. 27–38.

[18] Imatest. *Projective Camera Model*. Accessed: 2024-05-17. 2023. URL: https://www.imatest.com/support/docs/pre-5-2/geometric-calibration-deprecated/projective-camera/.

[19] Jagt. *clumsy, an utility for simulating broken network for Windows Vista and above*. Accessed: 2024-05-12. 2023. URL: https://jagt.github.io/clumsy/.

[20] C. Krautz. *What Are The Coordinates*. Accessed: 2024-05-17. 2023. URL: https://medium.com/@christophkrautz/what-are-the-coordinates-225f1ec0dd78.

[21] T. Kummarikuntla. *Camera Calibration with OpenCV*. Published in Analytics Vidhya. Retrieved from https://medium.com/analytics-vidhya/camera-calibration-with-opencv-f324679c6eb7. 2019. URL: https://medium.com/analytics-vidhya/camera-calibration-with-opencv-f324679c6eb7.

[22] K. LibreTexts. *Applications of Linear Interpolation and Extrapolation*. Accessed: 2024-05-20. 2023. URL: https://k12.libretexts.org/Bookshelves/Mathematics/Book%3A_Mathematics_(Grade_11)/4%3A_Linear_Equations_and_Inequalities/4.7%3A_Applications_of_Linear_Interpolation_and_Extrapolation.

[23] E. Malis and M. Vargas Villanueva. "Deeper understanding of the homography decomposition for vision-based control" (2007).

[24] M. Nabiyouni, S. Scerbo, D. A. Bowman, and T. Höllerer. "Relative effects of real-world and virtual-world latency on an augmented reality training task: an ar simulation experiment". *Frontiers in ICT* **3** (2017), p. 34.

[25] "Tracking methods in xr hmds". *arXiv* (2024). Accessed: 2024-05-30. URL: https://arxiv.org/abs/2201.13278.

[26] Unity Technologies. *Best Practice: Making Believable Visuals*. Accessed: May 14, 2024. 2020. URL: https://docs.unity3d.com/2020.1/Documentation/Manual/BestPracticeMakingBelievableVisuals1.html.

[27] *Valve Index Base Station*. Accessed: 2023-05-22. Steam, 2023. URL: https://store.steampowered.com/app/1059570/Valve_Index_Base_Station/.

[28] N. Van Oosterwyck. *Real Time Human Robot Interactions and Speed Control of a Robotic Arm for Collaborative Operations*. PhD thesis. May 2018. DOI: 10.13140/RG.2.2.28723.53286.

[29] *Varjo XR-3*. https://varjo.com/products/varjo-xr-3/. Accessed: [date].

[30] WebRTC. *Web Real-Time Communication*. Accessed: 2024-05-12. 2024. URL: https://webrtc.org/.

[31] T. Weikert et al. “Head motion prediction in xr” (2021).

[32] *XReal*. https://www.xreal.com/. Accessed: [date].

[33] Zach. *Aruco Markers*. Accessed: 2024-05-12. 2021. URL: https://fab.cba.mit.edu/classes/865.21/people/zach/arucomarkers.html.

[34] J. Zhang, H. Yu, H. Deng, Z. Chai, M. Ma, and X. Zhong. “A robust and rapid camera calibration method by one captured image”. *IEEE Transactions on Instrumentation and Measurement* **68**:10 (2018), pp. 4112–4121.

[35] Y. Zhang, J. Wang, Y. He, X. Ji, Y. Kang, D. Liu, and B. Li. “Furion: towards energy-efficient wifi offloading under link dynamics”. In: *2016 13th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*. 2016, pp. 1–9. DOI: 10.1109/SAHCN.2016.7732994.