Utilising Computer Vision to Create an Augmented Reality Training Aid for Basketball Shooting

A thesis presented in partial fulfilment of the
requirements for the degree of Master of Engineering at Massey
University, Manawatū, New Zealand.

Damian Clinton Hansen

2022

Abstract

The use of Augmented Reality (AR) technology can improve an athlete's capabilities and help physical trainers add value to their teaching. Recently, the addition of technology in sports has come in many forms, from wearable sensors to monitoring the user's health and fitness and mapping their surroundings to virtual environments, which provide opportunities for practicing sport in remote locations, in isolation, and with additional feedback not available in real world practice. The goals of this research were to design and examine the feasibility of an immersive, fun and interactive experience through which the user can shoot a real basketball in an AR environment where the real hoop or target is replaced by a virtual target.

The object tracking capabilities of HoloLens proved to be insufficient for ball tracking in this project due to its low capturing speed and limited accuracy at longer distances. As a result, the Intel RealSense D455 device was used with the HoloLens in an attempt to achieve better ball tracking and then effectively communicate the 3D position data to the HoloLens. The results section records the accuracy of tracking the static and dynamic centroid position of the ball in 3D space. The static results yielded accurate centroid values at 1m; however, beyond 2m, the noise affected the accuracy of the values. From the dynamic results, using two larger distinct regions of interest (ROIs) within 2m from the camera yielded the most accurate launch velocity predictions. UDP worked reliably to send this 3D data as strings to the HoloLens, which then transformed the 3D data using the ICP algorithm to calibrate the axes of the D455 and the HoloLens. The ICP algorithm had small inaccuracies when calibrating the two axes because it is commonly used to calibrate large data sets; however, all the mean error values between the transformed coordinates and known coordinates for all the axes fell within one standard deviation of the mean error. The final trajectory experiment was successful in spawning a ball at a 3D location and produced the most accurate results when hitting a target 0.8m to 1.2m away from the D455~~.~~, however a final trajectory prediction analysis incorporating the launch velocity error measured in this work clearly demonstrated the technology is not currently suitable for use in long distance applications such as a typical basketball free throw.

The current limitations of this tracking method and its efficiency as an AR training tool were explored and future recommendations were provided for better tracking accuracy in AR using the HoloLens.

Acknowledgements

- I would like to thank my supervisor Dr Gabe Redding for introducing me to this field of research and for his guidance and support throughout the duration of this project and for spending extra hours remotely assisting me with my experiments.
- Prof. Johan Potgieter for giving me the opportunity to do a masters through MAF Digital Labs and Massey.
- Acknowledgement to Mr Darrin Hurd for generously allowing me to use the HoloLens 2 for the experiments conducted in this project.
- I am grateful to my parents, brother and my fiancée for their support and encouragement whenever I needed it most throughout this project.
- Gratitude is expressed to Callaghan Innovation for the financial support provided.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

The technological era has seen a vast shift towards user entertainment and creating immersive user experiences. One such technology that has seen advancement in this area is augmented reality. AR, along with its interactive experience, is also proving useful in many different industries such as manufacturing, medicine, and sports where users can interact with virtual holograms within the real environment. In this thesis, the focus will be on how this technology can be used in sports, specifically as a basketball shooting training aid. This technology can improve athlete's capabilities and help physical trainers add value to their teaching. Recently, the addition of technology in sports has come in many forms, from wearable sensors to monitoring the user's health and fitness and mapping their surroundings to virtual environments which provide opportunities for practicing sport in remote locations. AR merges computer-generated elements with the real world through audio, video, graphics and location data to enhance the environment and situations and create perceptually rich experiences through an interactive and digitally manipulatable environment (Soltani & Morice, 2020).

Within this thesis, the HoloLens 2 will be used as a Head Mounted Display (HMD) through which a user will visualise a holographic display of a basketball environment. The Microsoft HoloLens 2 is one of the latest HMDs that facilitates mixed and AR applications and can project virtual holograms so that they appear from the user's perspective to occupy real physical space. All hardware is embedded within the device, which creates a compact HMD that can be used wirelessly and in multiple environments. One purpose of this project is to design and examine the feasibility of an immersive, fun and interactive experience through which the user can shoot a real basketball in an AR environment where the real hoop or target is replaced by a virtual target.

The required physical elements will be the basketball and a reasonably sized physical space for shooting, whereas the virtual environment will include a spawned virtual basketball and a virtual basketball hoop/target. Consequently, the physical ball, when thrown, will spawn into a virtual ball which will be seen through the HoloLens 2. As illustrated in **Figure 1.1**, the virtual ball's motion should be simulated so that it follows a trajectory that is as close as possible to that followed by the real ball. The virtual ball will interact with the virtual hoop/target, and this interaction should also be simulated to represent the desired physical interaction with the target (this may or may not be a real hoop depending on the application, and indeed there is the opportunity to have many and varied target interactions to create a wide range on interactive experiences).

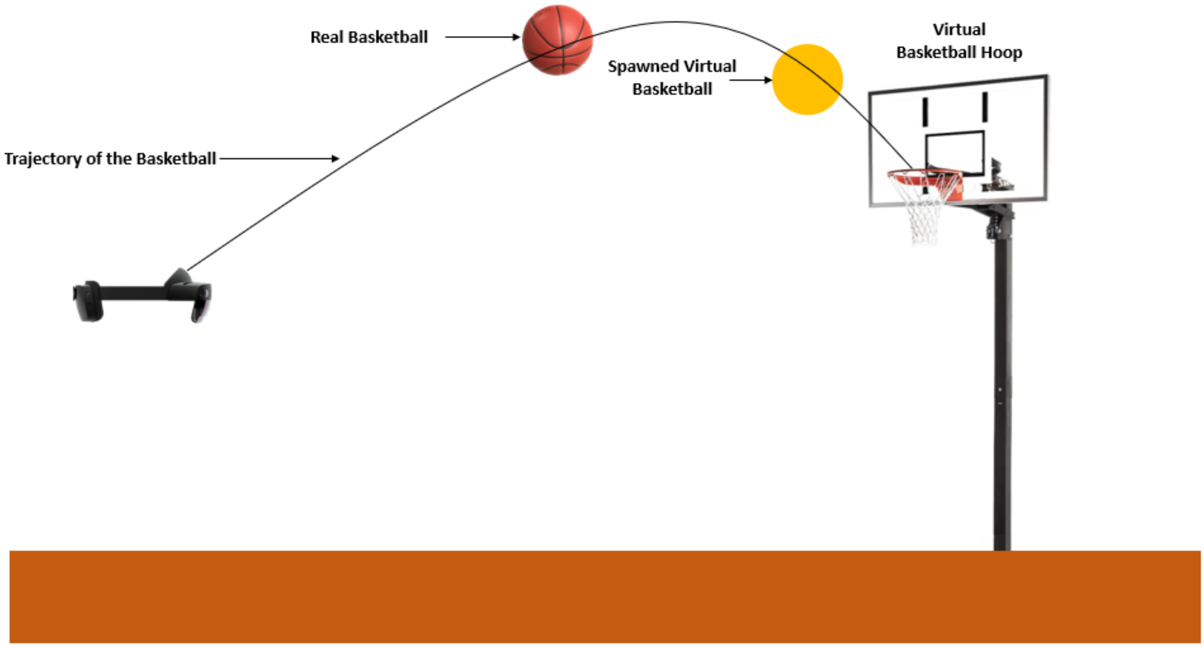

Figure 1.1. Basketball Training Aid Design System (Basketball hoop - (Spalding, 2022),HoloLens 2- (Microsoft, 2022d))

In this project one object, a basketball, will need to be tracked for as long as it takes to get sufficient information to simulate its trajectory at the level of accuracy desired. The two challenges envisaged in this work are (1) tracking the ball so that information sufficient to simulate its trajectory as accurately as possible, and (2) subsequent physical simulation of the ball target interaction. These two problems can be treated independently in that the 1st must clearly be solved before the 2nd for the 2nd part to be meaningful. Therefore, in the interests of setting up a feasible master's thesis, the scope of this work was narrowed to focus on solution to the 1st challenge, with the solution of the 2nd being relegated to the domain of future work.

There are two options for tracking the ball. The first is simply to track it all the way to the target, in which case no simulation is required. The 2nd is to track it for a period sufficient to obtain the information to accurately simulate it. Which is best, will likely depend on the capability of the hardware on the HoloLens. For this work, both the Unity and Unreal Engine game engines were seen as compatible 3D environments in which interactive AR scenes can be visualised with the HoloLens 2. **Error! Reference source not found.** will examine which of these options is the most suitable to solve the problem at hand. The commercial funder of this research has identified that ideally the onboard hardware of the HoloLens should be used for ball tracking and if this is not possible, the RealSense D455 depth sensor range can be explored. The pros and cons of the object tracking capabilities of both these systems are explored in detail in **Error! Reference source not found.**.

As a result, the two objectives of this thesis are (1) tracking the ball so that information sufficient to simulate its trajectory as accurately as possible, and (2) subsequent physical simulation of the ball target interaction. Therefore, in the interests of setting up a feasible master's thesis, the scope of this work was narrowed to focus on solution to the 1st challenge, with the solution of the 2nd being relegated to the domain of future work.

Chapter 2 Literature Review

Considering the problem that has been clearly described in Chapter 1, the literature needs to be surveyed to understand how the HoloLens 2 can be used as an AR object tracking tool. This chapter expands on how to achieve this by discussing how its hardware and software features can be used with a compatible 3D game engine to create a fun and immersive AR environment. As a result, different 3D object tracking techniques are explored in this chapter, including which tracking technique will suffice to obtain the desired objectives stated in Chapter 1. Thereafter, two popular 3D game engines are explored to determine which one will be best suited for the HoloLens 2. The chosen game engine will be used as a tool to simulate the physics of a ball in motion, including its trajectory and how it interacts within a 3D environment. All these topics will be compared with related research to decide which combination of hardware and software components are necessary to achieve the desired objectives stated in Chapter 1. Some of the methods of how the HoloLens can be used in various applications (discussed in Chapter 2.1 and Chapter 2.1.1) were outside the scope of this work and were instead meant to serve as additional information of the extended functionalities of the HoloLens 2.

2.1. Microsoft HoloLens 2

The desired purpose of this section is to understand the HoloLens 2 device and its capabilities. The HoloLens first generation device was introduced in 2016 (Hübner et al., 2020). Its capabilities included augmenting the physical environments of the user with holograms that are displayed through its transparent stereoscopic display unit. This device can be used as a standalone holographic computer; however, its true functionality is displayed when used in conjunction with a 3D application development environment such as Unity or Unreal Engine. Consequently, with these platforms, the user can then deploy virtual environments onto the device, creating augmented reality experiences which include 3D objects which appear to the user to be part of the real scene and can be manipulated within the augmented environment.

Along with its upgraded and improved HoloLens 2 counterpart, it is widely used in multiple industries such as industrial process management and engineering, surgery, education and even for human-robot interaction (Hübner et al., 2020). Despite its existing adoption in these fields, the general field of mixed reality is still new, but the HoloLens 2 creates opportunities for researchers to explore the fields of computer vision and robotics through its research mode, which provides direct access to key sensors that are part of the hardware. The HoloLens is widely used in many applications such as object detection, object tracking, and 3D scanning and can be used as a stand-along device or in combination with other devices. Farasin et al. (2020) uses a cloud system and the HoloLens system to perform real-time object detection by using deep convolutional neural networks (CNNs). Similarly, Bahri et al. (2019) created an object detection system using the YOLOv2 (You Only Look Once) deep convolutional neural network in a client-server configuration via the HoloLens to detect and recognize between objects in the real world. A real-time, indoor scanning application using the HoloLens to scan indoor environments and build 3D interactive scans is mentioned in Z. Wu et al. (2020). These systems used the HoloLens as a stand-alone device, whereas in Garon et al. (2016), the Intel RealSense Camera was used with a HoloLens to establish a real-time 3D object detection and pose estimation system. This was because the HoloLens device suffers from multiple limitations such as the lack of real-time, high quality depth data, which restricts their use as research tools. As a result, the RealSense device was connected to a stick personal computer (PC) via USB and mounted on the HoloLens headset. This PC then transferred high resolution depth information in real-time to the headset via Wi-Fi.

AR has benefitted the manufacturing and assembly industry through improvements in first time quality and reduced training times (Evans et al., 2017). A major application in AR technology is its use as a tracking tool and for most AR applications, some combination of wearable or external tracking systems are necessary (Zhou et al., 2008) who review the evolution of tracking research including sensor-based, vision-based and hybrid tracking techniques and state that computer vision methods accounted for over 80% of tracking research papers in International Symposium on Mixed and Augmented Reality (ISMAR). They go on to say that most recently, model-based tracking methods have become one of the dominant computer vision tracking techniques presented at ISMAR. An example of this tracking system used a CAD model which was created by hand and a piecewise parametric representation of complex objects such as straight lines, spheres and cylinders. Furthermore, Itoh et al. (2016) describes a tracking technique where they implement an augmented motion prediction system that integrates predictive information from an object in motion into a user's field of view (FoV), which enables the user to enhance their perception of its future path using information presented by the HMD. The section below details some of the latest examples of the use of AR within sports.

2.1.1. Recent use of Augmented Reality in Sports

Some of the many benefits of using AR in sports include learning sport skills. As mentioned in Soltani and Morice (2020), perceptual learning methods are advantageous for specific and robust training where learners are exposed to various situations to enhance the user's understanding of different domains. They also mention the benefit that AR plays through providing additional information. An example of this includes, increasing players' accuracy in estimating future ball predictions increased by three times through a vision augmentation system that could estimate objects' trajectories and predict their future positions. Furthermore, they showed that displaying real-time game-related information increased basketball players' awareness and positively influence their in-game decisions. In da Silva et al. (2021), they mention how the super positioning of objects helps in the dynamics of understanding the sport by showing details practically and intuitively employing images. This study explores the input and output interaction types in AR-designed systems and discusses how users can benefit from interaction with AR systems through various motion detection, camera aiming, GPS, rotational controls, and character movement controls and what the systems' responses are to the users.

2.1.2. Specifications

The Microsoft HoloLens 2 is an untethered holographic computer and is an improved version of the first-generation device. Its updates include a larger field of view, a custom deep neural network DNN core, fully articulated hand tracking and eye gaze tracking intended to provide a more comfortable and immersive experience together with more options for collaborating in mixed reality. Whoever wears this headset can move freely in real space and interact with various 3D models via hand gestures and voice assistance through the data displayed directly on the glasses. Its self-contained, ergonomic design is designed to fit the specific user's head shape and is equipped with two on-board processing units, sensing capabilities and audio that is captured with a 5-channel microphone array. A detailed outline of these specifications can be seen in **Table 2.1**.

Table 2.1. HoloLens 2 Device Specifications (Microsoft, 2022a)

Device Specifications		
Component	**Function**	**Description**
Display	Optics	See-through holographic lenses (waveguides)
	Holographic resolution	2k 3:2 light engines
	Holographic density	>2.5k radiants (light points per radian)
	Eye-based rendering	Display optimization for 3D eye position
Sensors	Head tracking	4 visible light cameras
	Eye tracking	2 Infrared (IR) cameras
	Depth	1-MP Time-of-Flight depth sensor
	Inertial Measurement Unit (IMU)	Accelerometer, gyroscope, magnetometer
	Camera	8-MP stills, 1080p30 video
Audio and Speech	Microphone array	5 channels
	Speakers	Built-in spatial sound
Compute and Connectivity	System on chip	Qualcomm Snapdragon 850 Compute Platform
	Holographic processing unit	Second-generation custom-built holographic processing unit
	Memory	4-GB LPDDR4x system DRAM

	Storage	64-GB UFS 2.1
	Wi-Fi	802.11ac 2x2
	Bluetooth	5.0
	USB	USB Type-C DRP
Power	Battery Life	2-3 hours of active use and up to two weeks of standby time.
	Battery technology	Lithium batteries
Fit	Sizing	Single size with adjustable band and fits over eyeglasses.
	Weight	566 grams

The headset hardware includes an adjustable visor and headband along with a variety of buttons on the device including brightness buttons, volume buttons and a power button. A USB-C port is also located on the device for charging and data transfer purposes. The Holographic Processing Unit (HPU) and Computer Processing Unit (CPU) make up the processing system on the device. The HPU (second-generation custom built) is on the front part of the device and enables low-power, real-time computer vision by running all its algorithms, such as head tracking, hand tracking, eye gaze tracking and spatial mapping. Whereas the CPU, fitted onto a System on Chip (SoC) Qualcomm SnapDragon 850, is on the rear of the device and runs the desired applications when needed. These processors manage the real-time update of displayed holograms and the operating system, which can determine its relative position with respect to the real world, therefore building a real-time map of the surrounding environment.

To achieve a satisfying AR experience, the device is equipped with a variety of sensors including a time-of-flight (ToF) depth sensor, a red, green and blue (RGB) camera, four tracking grayscale cameras, and an Inertial Measurement Unit (IMU) that allows real-time perception of nearby shapes and obstacles (Microsoft, 2022e). These features can be seen in **Figure 2.1** and **Figure 2.2**.

2.1.3. HoloLens SDK

The HoloLens follows the Universal Windows Platform (UWP) development framework and is used with the Mixed Reality Toolkit (MRTK) library to enable users to develop mixed reality environments (Microsoft, 2022, January 1). These development environments include:

1. Visual Studio - An integrated development environment for building and deploying UWP applications to the HoloLens.
2. Unity 3D – Primarily used to develop applications in 3D space. Additional components are inserted into a Unity scene along with 3D objects to design interactions and behaviours.
3. MRTK -Unity – a cross-platform development tool for mixed reality applications within Unity and comes with many functions, including:

a. Cross-platform input system and building blocks for spatial interactions and user interface (UI)

b. Allows users to see changes immediately through an in-editor simulation

c. Acts as an extensive framework for developers

d. Supports an extensive range of platforms.

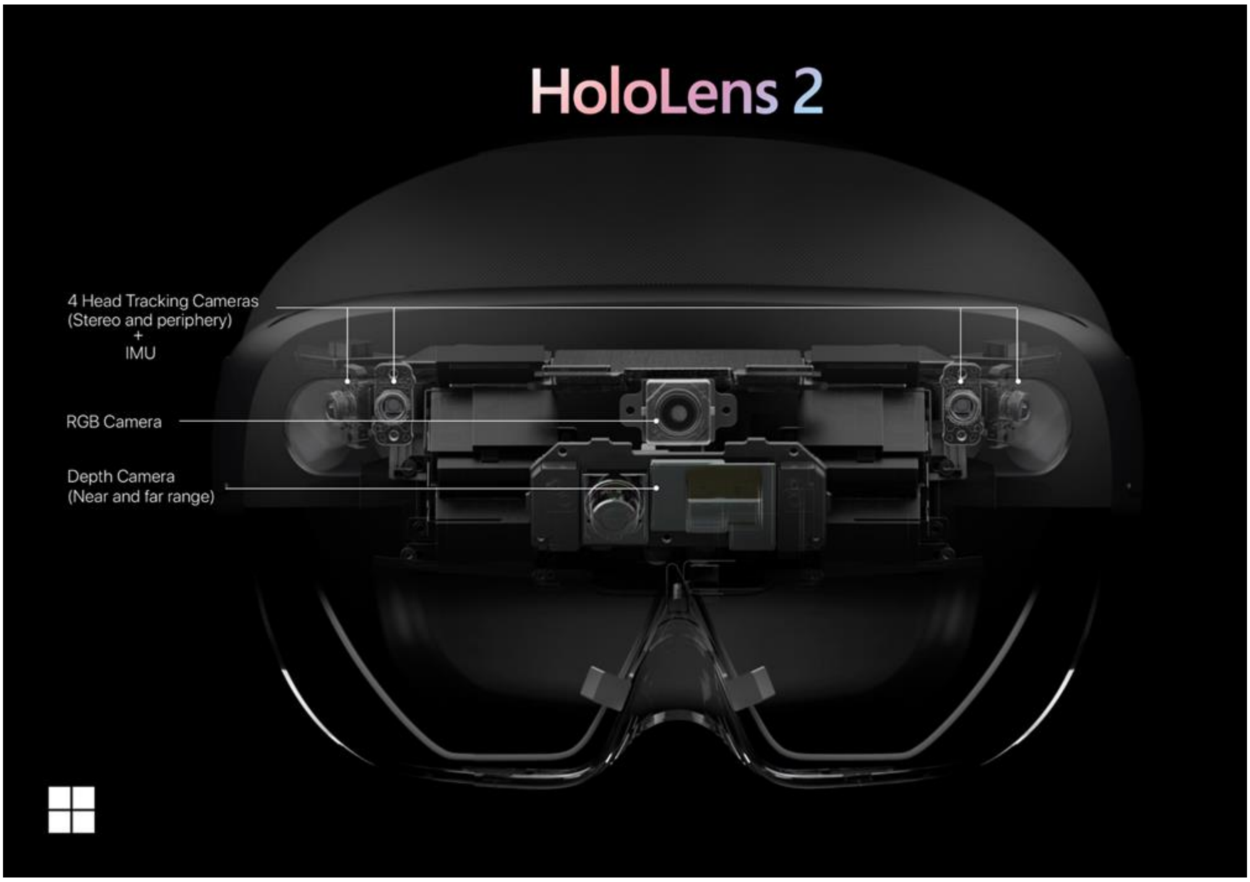

Figure 2.1. HoloLens 2 Tracking Features (Microsoft, 2022g)

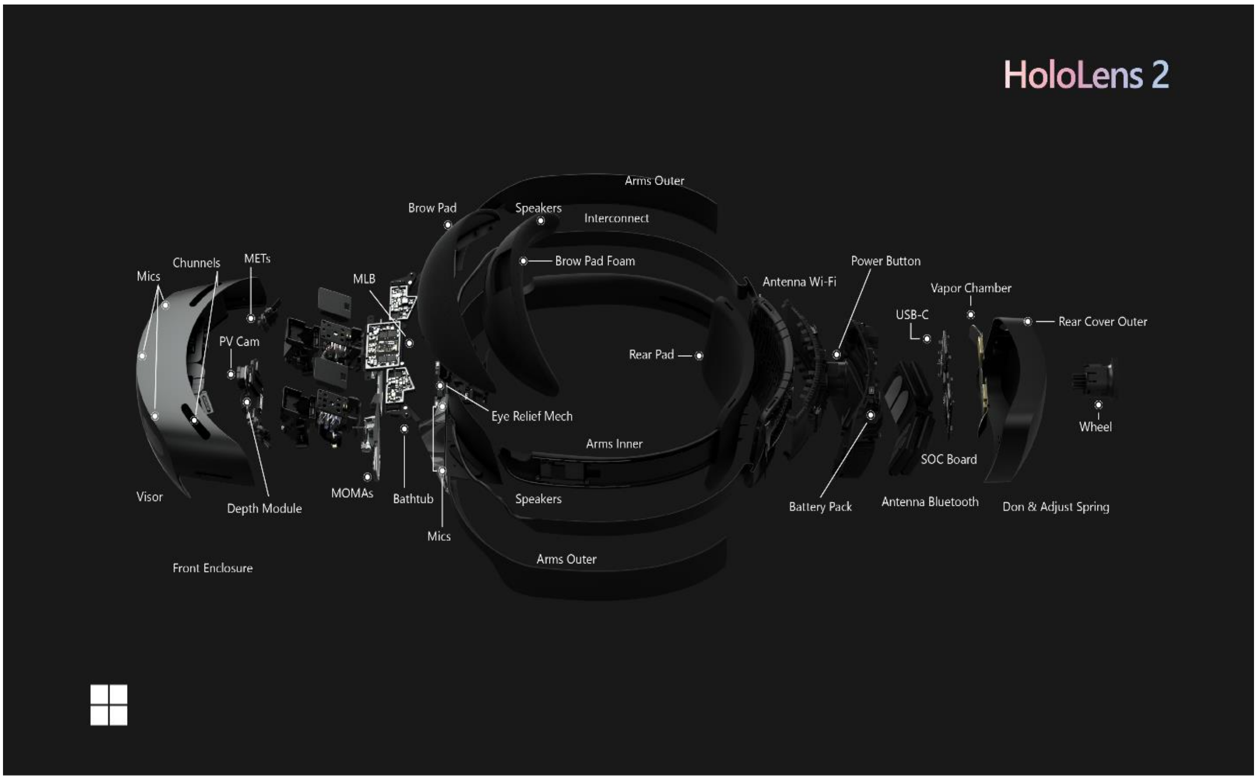

Figure 2.2. Components and Features of the HoloLens 2 (Microsoft, 2022f)

2.2. 3D Applications Environment

A 3D application environment is essential for the deployment of virtual 3D objects that are interactable, and this specific environment will need to be compatible with the HoloLens 2. In the specific 3D application to be developed in this work, a ball will be tracked for a sufficient period to obtain the information to accurately simulate its trajectory. There are two 3D game engines in which this is possible, namely Unity and Unreal Engine. Both environments are game development platforms and can be used cohesively with the HoloLens 2 device. This section will outline these compatible platforms to determine which one will suffice to simulate a ball in motion.

2.2.1. Unity

Unity is currently one of the most popular platforms for creating and operating interactive, real-time 3D content. It provides the tools to make advanced games and publish them to a wide range of devices and is a popular development tool for the Microsoft HoloLens 2. Its underlying runtime code is written in C++ and all its development scripting is created using C# in Microsoft Visual Studio. Microsoft provides extensive documentation on how to develop for the HoloLens using Unity, which makes it easy for users to access (Evans et al., 2017).

Additional to being a game development engine, it has also made progress in automotive, transportation and manufacturing, film, animation and cinematics and architecture, engineering and construction (Unity, 2021). As mentioned in 2.1.2, MRTK is used within Unity to develop mixed reality applications in 3D space that can be displayed onto the HoloLens. Components such as Spatial Understanding, Input Mapping, and the HoloLens Camera along with 3D objects are placed into a Unity game scene to design interactions and behaviours (Gupta et al., 2018). Unity is used in many AR applications, such as in Itoh et al. (2016), where it is used to create virtual objects and visualizations and then collect 3D position data for prediction and rendering of virtual information, which was then sent as image frames to the HMD. In Stark et al. (2020), the Unity engine is used as a graphical user interface in a mobile application to recognize and control mechatronic devices, in this case a joystick.

2.2.2. Unreal Engine

Unreal Engine 4 is a powerful open-source tool that supports mixed reality in C++ and Blueprints, and the development for head-mounted XR-devices and handheld AR device. The HoloLens 2 is also supported in Unreal Engine through OpenXR Application Programmable Interfaces (APIs). This gives users access to unique features, such as streaming and native deployment, finger tracking, gesture recognition, meshing, voice input, spatial anchor pinning, and more. Furthermore, this game engine incorporates the MRTK that provides Unreal users with the opportunity to develop applications with the HoloLens 2 (Microsoft, 2022i). Like Unity, users can also deliver interactive experiences and immersive virtual worlds. As mentioned above, both engines allow for the same features to be used on the HoloLens and enable the user to explore AR and Mixed Reality using the MRTK. Additionally, Unity does support world locking tools, azure object anchors and azure remote rendering, which is unsupported in Unreal Engine. However, both game engines support all the useful features needed when developing an XR application. However, Unity has supported the HoloLens since its first inception and due to its popularity and simple integration with the HoloLens 2, it was chosen over Unreal Engine as the sole 3D application development environment in this project.

2.3. Object Tracking

Object tracking is vitally important in computer vision applications (Swalaganata et al., 2018), such as human computer interaction, surveillance, medical imaging and is one of the fundamental enabling technologies for AR. At the core of object tracking, the initial position and size of a target in a video frame is given and thereafter an estimation of the trajectory and the subsequent states of the target is derived (Y. Wu et al., 2013). There are three basic steps when tracking a moving object. At the first step, the object must be detected. Thereafter, the selected object must be tracked and then the object's behaviour is identified through the analyses of the object’s motion.

There is extensive research being done in this field because of the various challenges that surround optimizing object tracking (Runze et al., 2017). These challenges include, tracking objects which are similar in appearance to their surroundings, objects that are situated in highly cluttered and poorly lit scenes or objects that are partially or completely occluded during object tracking.

Zhou et al. (2008) discusses some of the fundamental tracking techniques including, sensor-based, vision-based, and hybrid tracking. Sensor-based tracking techniques are based on sensors such as magnetic, acoustic, inertial, optical and/or mechanical sensors and were popular in early object tracking research. These tracking systems are compared to open loop systems whose output is perceived to have error. Conversely, vision-based tracking techniques correct errors dynamically by using image processing methods to calculate the camera pose relative to real world objects. They also show recent trends in visual tracking (computer vision) to be the most researched field in tracking techniques, specifically model-based tracking methods, which use a model of the features of tracked objects based on the distinguishable features. However, because of the advancement in recent AR technologies, more robust tracking methods have been required. Consequently, this has seen the importance in combining multiple sensing technologies to form hybrid tracking techniques.

There is a growing demand for tracking objects in sports games. One such method improved dynamic object tracking ability, including occasionally locating occluded balls, by taking consecutive frames and generating a heatmap that indicated the position of the object using CNN and deconvolutional neural network (D-CNN) (Huang, 2019). Furthermore, Moon et al. (2018) discusses how object tracking performance in sport video analyses can be enhanced by using background separation, object shape confirmation and edge map extraction due to its ability to detect moving objects. Another method used region-based analyses and retrieval to track objects in football scenes by repeatedly identifying them from the database using prior knowledge such as shirt colour, shorts colour and grass colour (Andrade et al., 2005). This technique partitioned the image into several smaller regions, such as size, colour, intensity, texture and neighbourhood. It also discusses how it is advantageous for both low-motion and high motion scenes as it segmented noise and performed better respectively.

Broadly defined, object tracking can be classified into two main categories namely, image-based and video-based object tracking. Image tracking is used to automatically recognize and track images and is mostly applied in the field of AR. Video tracking analyses the video frame sequentially and combines an objects past location with its present location using bounding boxes for prediction (Barla, 2022). Both these tracking techniques can be defined by single object tracking and multiple object tracking. However, for this project, the object will be a single basketball moving through the air towards a specific target (target box) and the aim will be to estimate the trajectory of the ball in the image plane as it moves around a scene. Therefore, single image tracking will be used for this project. There are multiple image tracking techniques that are available for computer vision applications. As the goal of this project is to track a moving object, RGB and RGB-D will be the two techniques explored in greater detail below.

2.3.1. RGB

Visual object tracking is one of the most researched areas in computer vision, with applications in automated surveillance, motion-based recognition, human-computer interaction and robotics (Asvadi et al., 2014). Conventional tracking processes emphasise the image formation between consecutive frames based on features such as colour, position, shape and velocity. These colour features are an important part of visual tracking since they are unchanged by translation, rotation and scale and are insensitive to partial occlusion and posture changes (Ku et al., 2015). Features like area, colour, speed and direction of motion are used in Rivlin et al. (2002) for tracking objects.

Typical colour-based methods are mean-shifts algorithms and the continuously adaptive mean-shift algorithms. However, colour-feature tracking does pose challenges in the natural environment due to occlusion and shadows. Furthermore, tracking drift and tracking failure can also occur due to background clustering, illumination and scale changes and can occur in traditional colour-feature algorithms such as mean-shift as they lack robustness. Ku et al. (2015) seems to reduce background clustering by using a method that calculates the histograms of an object's region and background. However, in a multiple camera scene, this method seemed to fail as the variable lighting caused the same object to be displayed differently in different cameras (Han & Kim, 2013). This problem was resolved using hue modelling to find the same object in multiple non-overlapping cameras.

2.3.2. RGB-Depth

Silvio Giancola et al. (2018) defines computer vision technology as using visual information to try and understand a given scene. The hardware on vision systems produces images or videos by measuring light intensity but can also be used to generate point clouds or meshes, whereas vision software focuses on detecting and tracking objects or reconstructing geometrical shapes. The integration of both the hardware and software components form 3D computer vision, which is essential in RGB-D tracking. Most existing 3D object tracking methods inherit 2D's experience and mainly rely on RGB-D information (Qi et al., 2020).

Simply put, an RGB-D frame is composed of a colour image and an aligned depth map (Slavcheva et al., 2017). Moreover, the RGB-Ds tracking capability incorporates depth information in addition to simple visual tracking used in RGB methods. For that reason, according to Liu et al. (2019), most RGB-D methods perform better than RGB methods. The RGB-D cameras can generate both a depth image and an RGB image through which a point cloud can be mapped. This comes about as each pixel of the RGB image is mapped into 3D space with the depth channel and the camera parameters. These 3D points encode both colour and 3D shape information of the scene.

RGB-D cameras make use of depth-sensing technologies including, active stereoscopy, structured-light, and time-of-flight (ToF). Structured light and active stereoscopy estimate depth using a triangulation process, whereas ToF uses direct depth measurement (Silvio Giancola et al., 2018). Active stereoscopy looks for artificially projected features from multiple cameras to reconstruct a 3D shape using triangulation and epi-polar geometry theory. The projected pattern serves as additional features with which to triangulate. Conversely, with structured light devices, a laser pattern is projected to the target to estimate the depth by triangulation. ToF systems use the time it takes for a light signal to go back and forth from a scene to estimate the depth. This is based on the principle of Light Detection and Ranging (LiDAR).

One of the earliest investigations into 3D object tracking using point clouds was used in S. Giancola et al. (2019) to test the versatility of shape completion for 3D object tracking in LIDAR point clouds using a Siamese tracker that encodes model and candidate shapes into a compact latest representation. Qi et al. (2020) further expands on 3D object tracking using point clouds through 3D template matching using Kalman filtering to generate 3D target proposals. Using only point clouds, this method achieves state-of-the-art results with deep learning and 3D target proposals. Primarily 3D object tracking in point clouds is used for applications in autonomous driving and robotic vision (Qi et al., 2020). The geometric point cloud-based approach has been influenced by stereographic, structured light and LIDAR sensors which are used abundantly in robotic perception. Many variants of the Iterative Closest Point (ICP) algorithm have been developed for location and tracking of objects in point clouds.

In Liu et al. (2019), a client-server architecture was used that enabled rapid calibration and real-time 3D dynamic scene rendering to develop a fast and robust method for RGB-D camera network calibration using a spherical object. Whereas Slavcheva et al. (2017) used RGB-D data to achieve dense 3D reconstruction of small-to-medium scale objects on the CPU and then refined initial pose estimates using an optimization framework. The Microsoft Kinect's depth images are used in Ding and Song (2015) and adapted to reduce model drift, handle occlusion and depth-based segmentation to find heavily occlude objects. Despite this success in 3D object tracking using RGB-D, it does come with some significant drawbacks. Firstly, it relies on visual cues which may fail if there is an illumination issue. Secondly, they also have no 3D tracking networks, thus limiting its representative power (Qi et al., 2020).

Based on the above-mentioned information, RGB tracking using a single camera will not suffice for this project as the ball will be moving in 3D space and will require depth information to identify its distance from the camera and its distance to the target area. Moreover, RGB object tracking methods use algorithms to identify features in objects prior to tracking them. However, this project does not require feature detection as only one object will be in the camera's FoV at any given time, hence there need not be a requirement to distinguish it from other objects or features in the scene (other than any potentially variable background). As a result, RGB tracking would be a rather cumbersome method to use in this regard, therefore, RGB-D tracking will be used as the sole tracking technique for this project. The depth-tracking capabilities of the HoloLens 2 such as its frames per second (fps), are explored in greater detail below to see how it fares in tracking a moving ball with the desired accuracy.

2.3.3. HoloLens 2

The HoloLens 2 was the most obvious choice for object tracking in this project as it eliminates the need for any registration in this work, or indeed any additional hardware. Its research mode allows users to have access to visible light environment tracking cameras, a depth camera, and multiple versions of the IR-reflectivity stream. Research mode is intended for academic and industrial researchers to explore new ideas in Computer Vision and Robotics and not freely available through the Microsoft Store or other distribution channels. A matter of interest for this project was the functionality of the depth camera and how it would fare in real-time ball tracking. Upon further investigation, it is seen that the depth camera operates in two modes namely, high-frequency (45 FPS) mode for near-depth and for hand tracking and a long throw, low-frequency (1-5 FPS) mode for far-depth sensing used by spatial mapping (Microsoft, 2022h). Specific distances for these modes are not specified in the reference papers and hence, these distances can be intuitively assumed based on the abovementioned reference points (hand tracking and spatial mapping). Enabling Research Mode has limitations such as higher battery usage and lower overall security and because depth data can only be accessed in this mode, it limits what the user can access. Additionally, the low frame rate will most likely hinder the ability to capture all the required frames when tracking moving objects. Despite its depth tracking limitations, it can be used in addition to other depth tracking devices for the purposes of this project. Garon et al. (2016) for example, bypassed this limitation by sending 3D data from a RealSense depth camera to the HoloLens using the User Datagram Protocol (UDP).

Based on the abovementioned evidence, the HoloLens 2 device will not suffice as a stand-alone object tracking tool and will need to be paired with an additional tracking device. In terms of lightweight relatively inexpensive systems, the two state of the art devices that are commonly used in computer vision and depth tracking are the Azure Kinect and the Intel RealSense Depth camera. These two specific 3D imaging devices are discussed and analysed below. They both use two of the main 3D imaging technologies discussed above – Active Stereoscopy and ToF respectively, which will be useful in obtaining valuable experimental 3D imaging data. The Azure Kinect is the upgraded form of the Kinect V2 used as the camera sensing technology in the Xbox motion tracking system and Intel RealSense has a range of depth cameras that could be used for ball tracking. These two devices are discussed in 2.3.4 below.

2.3.4. Intel RealSense D455

The Intel RealSense cameras are widely used in the fields of hand tracking, fine face recognition, machine vision and 3D scanning. The use of stereoscopic vision helps robots better understand 3D space, objects and shapes and enable them to move around in a 3D world (Kamel, 2021). Over the years, Intel has released various vision modules based on active stereoscopy and structured-light technologies from the SR300 camera module designed for short-range 3D scanning of small objects to Intel's D400 series with better depth resolutions. The latest D400 series cameras include RGB-D modules and image processing hardware, providing users with a flexible way to implement embedded 3D vision solutions (Silvio Giancola et al., 2018).

The main components of the Intel RealSense Depth Camera D455 is operated by its Vision Processor D4 and a depth module D450 and can be used in both indoor and outdoor environments. Stereo vision is implemented using a left imager, a right imager, and an optional infrared projector. In scenes with low texture, the depth accuracy is enhanced through the infrared projector emitting an invisible static IR pattern. Depth values are captured and calculated by the Vision Processor for each pixel by correlating points on the left image to the right image. In the end, a depth video stream is created through multiple depth frames created through processing depth pixel values (Kamel, 2021). The Intel RealSense D455 depth camera is illustrated in **Figure 2.3** and its specifications are detailed in **Table 2.2**.

Table 2.2. Intel RealSense Depth Camera D455 specifications

Function	Description
Depth Technology	Active Stereo
Image Sensor Technology	Global Shutter
Depth FoV (H x V)	87 x 58
Depth Resolution	Up to 1280 x 720
Depth Accuracy	<2% at $4m^2$
Depth Frame Rate	Up to 90 fps
RGB Sensor Technology	Global Shutter
RGB Sensor Resolution	1 MP
RGB Frame Rate and Resolution	1280 x 800 at 30 fps
RGB Sensor FoV (H x V)	90 x 65
Ideal Range (m)	0.6m to 6m

As is evident from **Table 2.2**, the RealSense D455 has quality depth tracking capabilities and would be suitable to use as a depth tracking tool for this work. However, because it uses active stereo, its error in depth tracking increases significantly with distance, specifically at distances beyond 2m (discussed in detail in Chapter 4.1.5) (RealSense, 2021) .

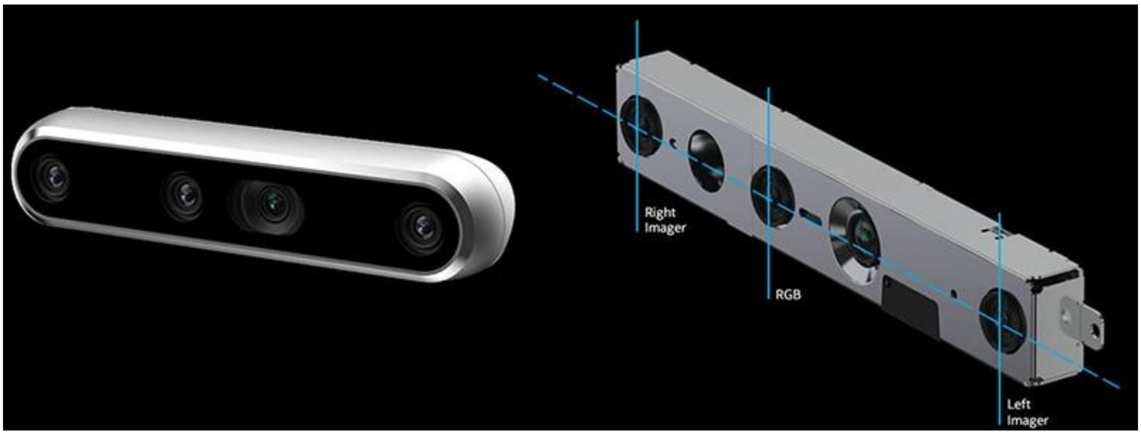

Figure 2.3. Intel® RealSense™ Depth Camera D455 (Source: Mouser Electronics) (Kamel, 2021)

The Intel RealSense SDK 2.0 is a cross-platform library for the RealSense depth cameras. It provides intrinsic and extrinsic information, depth, and colour streaming, as well as synthetic streams and is supported in multiple different programming languages.

2.3.5. Azure Kinect

The Azure Kinect developer kit (DK) is the latest Kinect version with voice and machine vision models, advanced AI sensors, and a variety of compatible SDKs that can be connected to Azure Cognitive Services. It is used for harnessing spatial data in multiple industries including manufacturing, retail, healthcare and media (Microsoft, 2022c).

The Azure Kinect DK integrates a 1-Megapixel (MP) ToF depth camera that supports multiple modes namely wide and narrow field-of-views (WFoV and NVoF respectively) and passive IR. Moreover, it also contains a microphone array with a video camera, and an orientation sensor with multiple modes, options and SDKs amongst other features as seen in **Table 2.3**and **Figure 2.4**.

Table 2.3. List of Azure Kinect Hardware Features (Microsoft, 2022b)

FEATURE		AZURE KINECT DK
Audio	Details	7-mic circular array
Motion sensor	Details	3-axis accelerometer + 3-axis gyro
RGB Camera	Details	3840 x 2160 px @30 fps
Depth Camera	Method	Time-of-Flight
	Resolution/FOV	640 x 576 px @30 fps
		512 x 512 px @30 fps
		1024x1024 px @15 fps
Connectivity	Data	USB3.1 gen 1 with Type-C connector
	Power	External PSU or USB-C
	Synchronization	RGB & Depth and IMU internal, external device-to-device
Mechanical	Dimensions	103 x 39 x 126 mm
	Mass	440 g
	Mounting	One ¼-20 UNC Four internal screw points

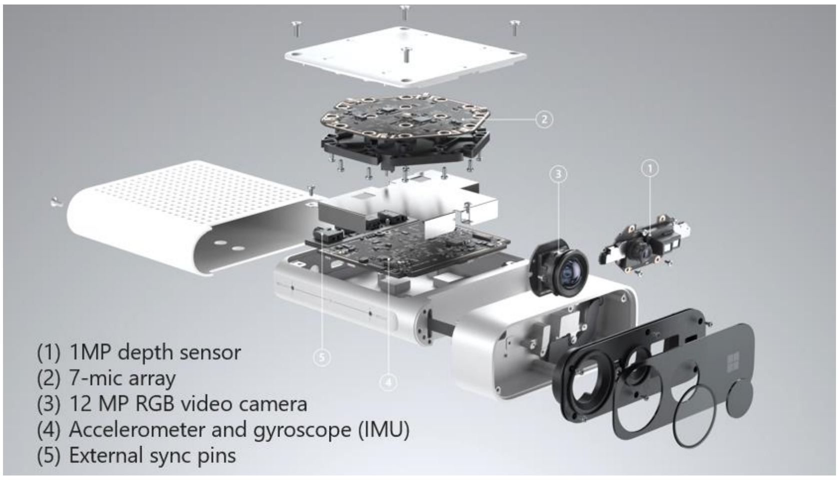

Figure 2.4. Hardware Specifications of the Azure Kinect DK Camera (Microsoft, 2022b)

The Azure Kinect DK development environment consists of multiple SDKs:

- Sensor SDK for low-level sensor and device access.
- Body Tracking SDK for tracking bodies in 3D.
- Speech Cognitive Services SDK for enabling microphone access and Azure cloud-based speech services.

The Kinect was used with the HoloLens in Fukuda et al. (2018) to generate a point cloud stream, which was then transferred to a connected PC in real-time. In this system, the Kinect for Windows SDK 2.0 was used as a tool to receive the point cloud stream via Wi-Fi (TCP/IP).

As explored above, both the Azure Kinect and the Intel RealSense D455 are used extensively in depth-tracking applications. Their ability to track moving objects over reasonably long ranges, including their ability to be paired with the HoloLens 2, make them a useful depth-tracking tool for this project. At the time this thesis was written, the Intel RealSense D455 was used as the preferred device as it was readily available and, as mentioned in Chapter 2.3.4, had been previously used with the first-generation HoloLens to improve the resolution of the depth data. Whilst the Azure Kinect would have been an equally justifiable choice for this project, it was out of stock during the period in which this work was carried out.

2.4. Simulation of basketball trajectory

Due to the limited range and fall off in accuracy with distance of the RealSense (briefly discussed in Chapter 2.3.4), the trajectory of the ball will need to be simulated rather than tracked all the way to the target. There are a few ways in which the camera can be placed to best achieve this goal. Firstly, it can be placed off to the side of the target or it can be placed facing the target. Additionally, it can be placed at the target hoop to track objects, which allows greater accuracy but runs the risk of getting hit by the ball. However, in this thesis, a target area was chosen instead of a basketball hoop and therefore this method was ruled out as an option.

2.4.1. Free throw trajectory in basketball

In basketball, the free throw is widely assumed to be one of the easiest shots. However, in the 2003-2004 NBA regular season, the highest free throw percentage was 92.7%, while some players averaged less than 50% (Okubo & Hubbard, 2006). Due to the importance of the free throw to the outcome of basketball games, there has been extensive research conducted into multiple factors that may have the greatest impact on free throws from the speed of the ball to the throw angle and kinematic parameters as well specific research into the simulation of the basketball shot trajectory.

In a basketball free throw trajectory, the centre of mass of a basketball follows a parabolic arc and can be influenced by multiple factors namely, distance from the hoop, different velocities and the initial angle of a free throw (Przednowek et al., 2018). Przednowek et al. (2018) used Particle Swarm Optimization (PSO) Algorithm to calculate the 2D coordinates of the ball trajectory by analysing ten parameters at three points of the throw. These included: velocities in three points, angles of the moving ball, height parameters and distance parameters. In Hamilton and Reinschmidt (1997), parameters such as ball-hoop interaction, ball elasticity, angle-speed-spin combinations, optimal trajectory and height of release were analysed. During their analyses, they neglected air resistance (drag) as this had a negligible effect on the free throws. Related research states that most long basketball shots have launch speeds of approximately 15 m/s (54kph) and can have drag forces as much as half the ball’s weight and lift forces of approximately 10% of the ball’s weight. Whereas, the more common two-point shots have smaller launch speeds and consequently smaller drag and lift forces (John Eric Goff, 2013).

Ultimately, the success of a shooter is dependent on two factors, namely the players understanding of the desired shot and the player’s consistency (Tran & Silverberg, 2008). The results showed that at a launch angle of 52 degrees the free throw was less sensitive to errors for a player with an average height of 1.981m and a release height of 2.133m. Furthermore, K. Hung and Macari Pallis (2004) used video, accelerometer, and eye movement measurements to demonstrate and verify the optimal release angle for a free throw is between 51 and 56 degrees and the importance of bent elbows and knees, and a snap of the wrist at the point of release, to improve the success of a free throw. However, the existence of human motor control limitations means that the optimal parameter combinations cannot be repeated exactly. Hamilton and Reinschmidt (1997) concluded that the most critical factor is probably the height of release, and the optimal trajectory requires a release with high backspin.

Based on the factors discussed above, drag had the least significant effect on the trajectory of a ball. Therefore, simple equations of motion were used to model the trajectory of the ball and the Unity game engine was used to take care of this.

Chapter 3 Methodology

Based on the literature review, there were a few gaps that were identified that are explored in more detail in this section. It was evident that the HoloLens' object tracking was not sufficient to be used as a tracking tool for the desired outcome of this project. Moreover, the depth-imaging tracking was the preferred option over RGB tracking as it did not require referencing objects in the scene before tracking could be performed. Therefore, the D455 was chosen as the preferred depth-tracking tool to identify the 3D location of the ball at a point in 3D space and then communicate this position data to the HoloLens. As a result, this section will explore what experiments were required to quantify the collected 3D position data from the D455 and how to effectively send that data to the HoloLens.

This section details the five experiments that were conducted to test the accuracy of the RealSense D455 and its compatibility with the HoloLens 2 in creating a basketball shooting aid device. Because the HoloLens 2 will be the only HoloLens used in this thesis, it will be referred to as HoloLens in the body of this thesis. The first two experiments were conducted using the D455 and the RealSense Software Development Kit (SDK) within C++ in Visual Studio. They included testing the accuracy of the camera when capturing 3D coordinates of both a static ball at various distances and a moving ball in all three axes. The third test utilised both the D455 and the HoloLens to test the effectiveness of the communication between the two devices and ensuring their compatibility. Thereafter, the fourth test included the aligning and registration of the axes (coordinate systems) of the two environments (RealSense and Unity 3D/HoloLens) and determining its accuracy. The final test tested how accurate the simulated virtually spawned ball was at hitting target areas placed at different distances from the D455.

The hardware components and software needed for these experiments are outlined below:

- Intel RealSense D455
- PC
- Basketball
- HoloLens
- Visual Studio C++ with RealSense SDK
- Unity 3D with Mixed Reality Toolkit (MRTK)

The common environmental factors that influenced the accuracy of the captured 3D coordinate data included:

1. Lighting
2. Object material
3. Distance of the object from the camera
4. Shape of the object

All these factors were considered in the static and dynamic experiments to test the effectiveness of the D455 in capturing 3D coordinate data. All the static experiments were conducted indoors in an office-lit environment, whereas the dynamic experiments were conducted both indoors and outdoors to test the effect that different environments have on tracking. The software development kit (SDK) of the RealSense D455 provided a wide range of options to alter the camera settings (intensity of projected light) to adapt to various environments (indoor or outdoor environment). However, for the static and dynamic experiments, only the default pre-sets were used. The object material and the object shape remained the same for both the static and dynamic experiments, whereas the distance variable altered according to the type of experiment to test the depth capturing functionality of the D455.

3.1. Static Experiments

For the static experiments, a ball as used as the test object and these experiments were conducted indoors at various distances to compare the difference in accuracy (if any) at one metre interval distances from one metre up to and including four metres. A detailed experimental layout is discussed below:

The required hardware components included:

- Intel RealSense D455 Camera with USB connection
- Laptop
- Basketball
- Table for standing the basketball and the camera

Software components:

- RealSense SDK (using C++ in Visual Studio)
- Intel RealSense Viewer

In this section, a set of 3D coordinate data points were captured, analysed, and compared with ground truth data. Ground truth data for the z axis was measured with a tape measure (**Figure 3.7**(b)), whereas for the x and y axes, ground truth data was taken using the RealSense axis supplied on the Intel RealSense Viewer Tool (**Figure 3.7**(a)). In Marin et al. (2019), depth ground truth data was captured using a line laser and a ToF camera to ensure accuracy and to use an external device instead of the device itself. However, for these experiments, a line laser was an expensive addition and could not be obtained. For each metre, the experimental data error for the z axis was compared with both the RealSense z centroid (ground truth measurement taken from RealSense Viewer) and the actual z centroid (ground truth measurement taken with a tape measure).

For each distance, the following data was collected and analysed:

1. *x*, *y* and *z* coordinates at each camera distance (1m, 2m, 3m and 4m)
2. For each axis, a comparison was made between the unfiltered, filtered (at 3 standard deviations) and measured centroids from the following two conditions:
    a. Two sample sizes, each containing 200 and 1000 3D coordinates respectively
    b. At each sample size, a set of four random 3D coordinates (one centroid iteration) were collected and the centroid was calculated. A variable (centroid iterations) was instantiated to iterate through several different centroid sets from each sample size.

Multiple sample sets were used to test the effect of having more 3D data points when calculating the centroid of the basketball. For the static testing, only two sample sets (200 and 1000 3D coordinate points) were analysed to assess the impact of large versus small amounts of 3D data coordinates. The centroid iteration variable was important as it allowed more randomly generated 3D coordinate sets to be used in the centroid calculation. Ultimately, resulting in a more accurate average centroid calculation (Chapter 3.1.2). For this reason, five different centroid iteration sizes (200, 400, 600, 800 and 1000) were used.

3.1.1. Experimental Outline

- Place RealSense D455 on flat surface facing the ball (**Figure 3.2**).
- Execute the script which captures 3D coordinates within the ROI (Appendix A, **Figure A 1**).
- Within this ROI, only the surface coordinates of the ball will be captured (**Figure 3.6**).
    i. The script used an algorithm that calculates the centroid of the ball based on a collection of 4 random 3D coordinates. This calculation is outlined in
    ii. Centroid Calculation.
- Within this script, the following methodology is followed:
    i. Create a ROI (
    ii. **Figure 3.*3***)
    iii. If object is in ROI, then collect all possible surface 3D coordinates.
    iv. Take 4 points multiple times over each sample set. **Figure 3.1** outlines this method.
    v. For each sample set, the centroid is calculated using determinant matrices (outlined in Chapter 3.1.2 below.

- Use the same data and filter the data points outside three standard deviations and compare it with the unfiltered data giving the representation of variability.

Figure 3.1 below shows an example of the workflow using a sample centroid iteration size of 200. This same process was repeated for the other four centroid iteration sizes defined above (400, 600, 800 and 1000).

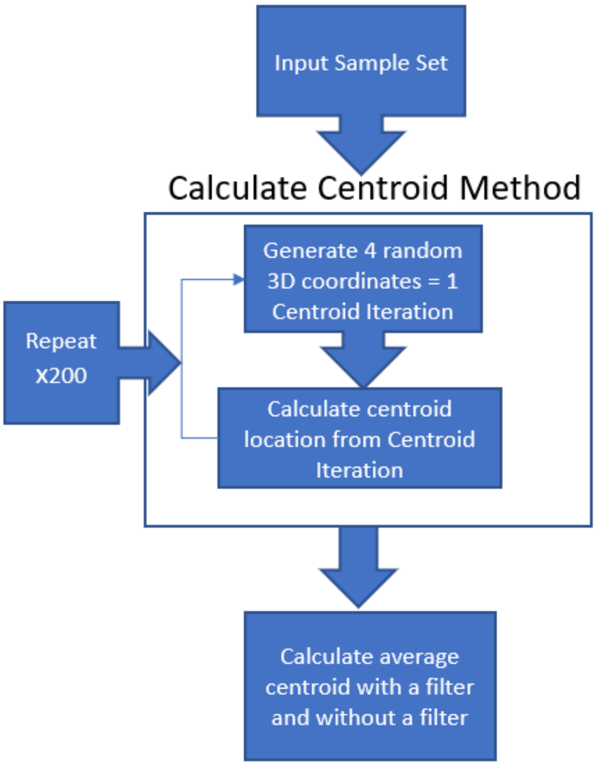

Figure 3.1. Diagram showing the process of how to get the final centroid location before and after filtering

In **Figure 3.1**, given a sample set of 3D points (200 or 1000) and a centroid iteration size = 200:

- A random set of 4 3D points are chosen for which the centroid is calculated. This process is repeated 200 times.
- The final centroid is the average of this centroid iteration size.
- Three standard deviations is used as a filter to eliminate outlying or noisy points and compared with the unfiltered average centroid value.

Figure 3.2. Side view of actual static experiment setup

The D455 was placed on a box facing the ball and the ball was balanced on a jar to line up the camera's axes with the centre of the ball as best as possible.

Figure 3.3 depicts the layout of the captured camera frame and the experimental setup seen from the front view.

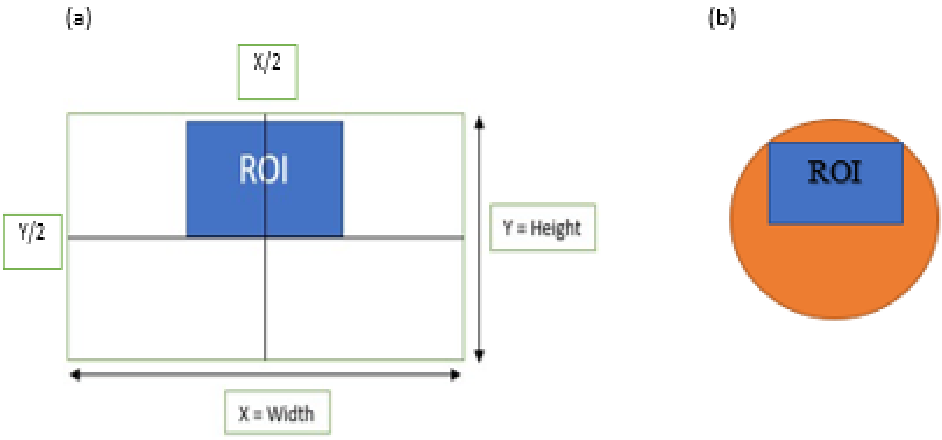

Figure 3.3. Front View Layout of the Captured ROI (a) Without the ball (b) On the ball (not drawn to scale)

The ROI is the area in which 3D coordinate data is captured at each metre. The front view details the width and height (x and y pixels respectively) that the camera's FoV captures. This area was defined to only capture a section area of the ball to reduce the probability of capturing outlier 3D coordinates as seen in

Figure 3.3(b).

The relationship between a stream's 2D and 3D coordinate systems is described by intrinsic parameters. Knowing the intrinsic parameters of an image allows two fundamental mapping operations to be carried out namely, projection and deprojection (Intel, 2019). Projection takes a point from a stream's 3D coordinate space and maps it to a 2D pixel location on that stream's images. Conversely, deprojection takes a 2D pixel location on a stream's images, specified in metres, and maps a 3D point location within the stream's associated 3D coordinate space, which will be used for all the following experiments. These two coordinate spaces are illustrated in **Figure 3.4**.

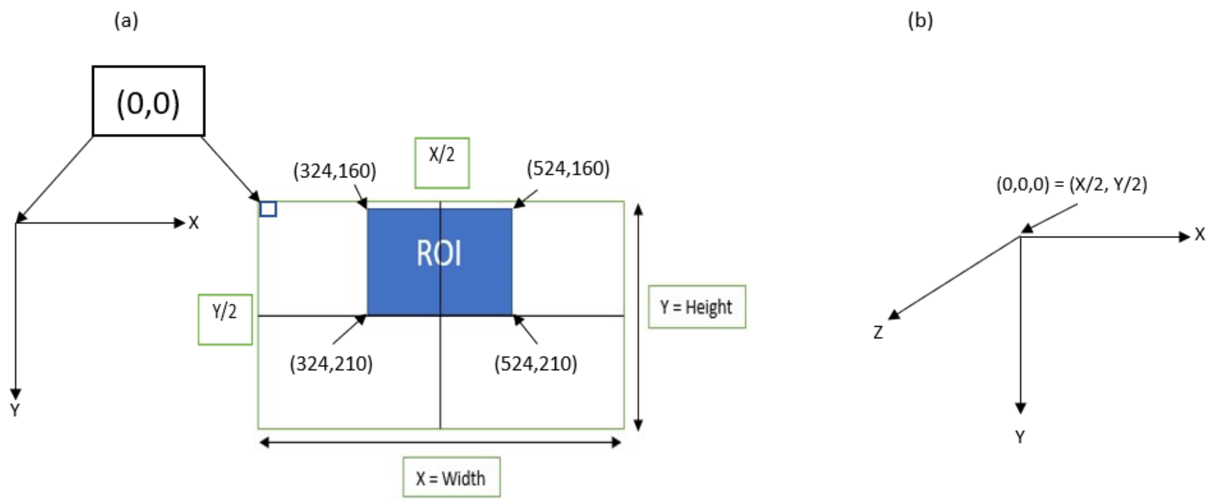

Figure 3.4. Coordinates in two 2D and 3D coordinate space (a) Pixel coordinates (b) Point coordinates

In **Figure 3.4**:

- **Captured camera frame:**

The origin of pixel coordinate frame is the top left-hand corner (0,0).

X = 848 (width of camera frame in pixels)

Y = 420 (height of camera frame in pixels)

- **Desired Region of Interest (ROI) in pixels** (Illustration not drawn to scale):

[X/2 – 100, X/2 + 100] = X [324,524]

[Y/2 – 50, Y/2] = Y [160,210]

For this experiment, the desired ROI was chosen so that the ball can be centred in-line with the camera and only its surface coordinates captured within this region. While both imagers were able to look at the range of y pixels in the image, the left imager looked at pixels ranging from x = 0 to x = 424 (x/2), whereas the right imager looked at pixels from x = 424 to x = 848. These pixels were then converted into world (point) coordinates using the abovementioned deprojection method.

Figure 3.5 gives an example of how the D455 takes these pixel coordinates and converts it into world coordinates.

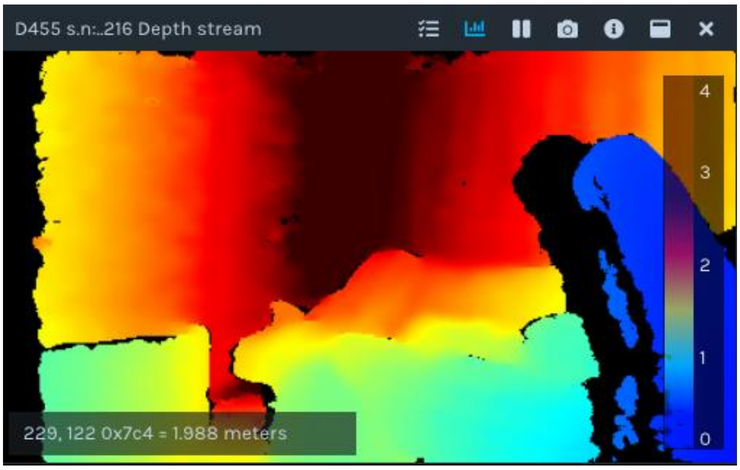

Figure 3.5. Example of a depth stream from the left imager of D455 illustrating the conversion of pixel coordinates into point coordinates (bottom left-hand corner)

In this case, (229,122) is the (*x*, *y*) pixel that is converted to a depth distance of 1.988m. From these 2D pixels, the *z* distance can be determined. The method with which this is determined is illustrated in **Figure 3.6**.

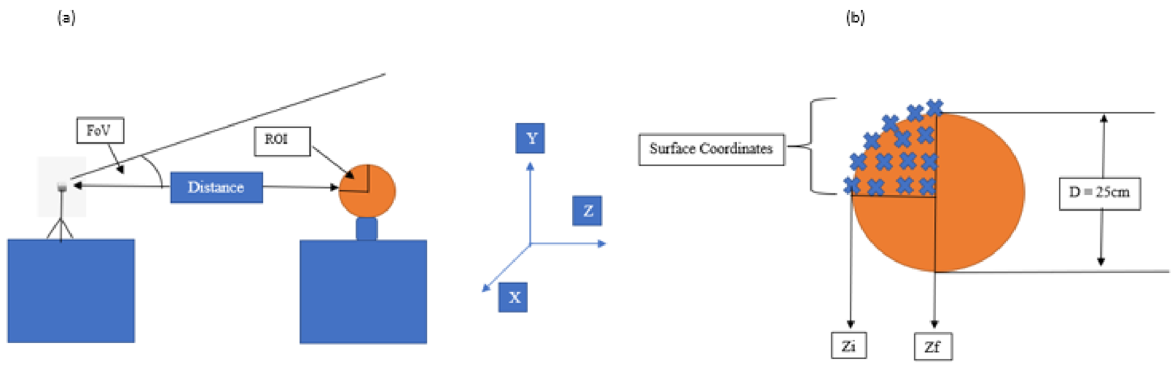

Figure 3.6. Side View of Experimental Layout (a) Zoomed out view (b) Captured surface within the ROI (zoomed in view)

In **Figure 3.6**:

- Place ball at various distances from the camera (1m, 2m, 3m, 4m) within the ROI that is required for each distance.
- The ball and the camera were set at the same height above the ground (approximately 1m) to ensure that the desired ROI of the ball was directly in front of the camera and the centre of the camera's coordinate axis was approximately in line with the centre of the ball.

- D is the diameter of the ball, and the ground truth measurement was taken to the front surface of the ball (Zi), therefore the radius of the ball (12.5 cm) was added to give the correct centroid distance.
- At each distance, the *z* distance of the ROI (Zi to Zf) was set to collect *z* coordinates within a 12.5cm range to reduce the possibility of collecting external 3D coordinates from other objects and to capture relevant depth coordinates from the surface of the ball.

3.1.2. Centroid Calculation

The method below details the equations used to calculate the centroid of a sphere. Bourke (2002) states that given 4 points in 3 dimensional space [(x_1,y_1,z_1) (x_2,y_2,z_2) (x_3,y_3,z_3) (x_4,y_4,z_4)], the equation of the sphere with those points on the surface is found by solving the following determinant.

$$\begin{vmatrix} x_1^2+y_1^2+z_1^2 & x_1 & y_1 & z_1 \\ x_2^2+y_2^2+z_2^2 & x_2 & y_2 & z_2 \\ x_3^2+y_3^2+z_3^2 & x_3 & y_3 & z_3 \\ x_4^2+y_4^2+z_4^2 & x_4 & y_4 & z_4 \end{vmatrix} = 0 \qquad (1)$$

There are two conditions that make the 4 points undefined:

1. No three combinations of the 4 points can be collinear.
2. All 4 points cannot lie on the same plane (coplanar)

Using expansion by minors (M_{1j}), the equation simplifies to the following:

$$(x^2+y^2+z^2)M_{11} - x \cdot M_{12} + y \cdot M_{13} - z \cdot M_{14} + M_{15} = 0 \qquad (2)$$

The general equation of a sphere with radius *r* centred at (x_0,y_0,z_0) is:

$$(x-x_0)^2 + (y-y_0)^2 + (z-z_0)^2 = r^2 \qquad (3)$$

Equating the terms from **(*3*)** allows one to solve for the centre and radius of the sphere, namely:

$$x_0 = 0.5 \cdot \frac{M_{12}}{M_{11}}$$

$$y_0 = -0.5 \cdot \frac{M_{13}}{M_{11}} \qquad (4)$$

$$z_0 = 0.5 \cdot \frac{M_{14}}{M_{11}}$$

$$r^2 = (x_0^2 + y_0^2 + z_0^2) - \frac{M_{15}}{M_{11}}$$

Note that these cannot be solved for M_{11} equal to zero. This corresponds to no quadratic terms (x^2, y^2, z^2) in which case we are not dealing with a sphere and the points are either coplanar or three are collinear. The code implemented to execute this calculation can be found in (Appendix A, **Figure A 2**).

According to (RealSense, 2021), the D400 stereo depth performs performs better indoors than outdoors. The quality of the ambient light can be determined with the use of the Depth Quality Tool through its default~~s~~ setting which shows a 3D point cloud with visual overlay. Hence, why this experiment was conducted indoors and in a well-lit room (**Figure 3.7**).

According to (RealSense, 2021), the ideal target should be placed between 50cm and 2m away from the camera and be parallel to the camera imaging plane. However, for this experiment the accuracy of the depth camera (with distance) is tested by performing additional experiments outside this ideal range. Because the D455 uses active stereo, as the ball moves further away from the camera the depth accuracy decreases significantly. Hence why various distances from 1m to 4m were chosen. For distances greater than 2m, the ROI needed to be reduced because as the object moves further away the object gets smaller and, if the ROI stays the same, there is more of a chance of collecting noisy or unwanted 3D coordinates. The position of the ball was placed so that the centre of the camera's 3D axes (taken from the Intel RealSense Viewer Tool) lies at the centre of the surface of the ball (**Figure 3.7**).

Figure 3.7. View of the coordinate axes of the D455 using the (a) Intel RealSense Viewer and (b) Tape Measure

This was done at each distance, so that the x and y coordinates of the ball would be the same (0 and 0 respectively). These coordinates were taken as ground truth data from the Intel RealSense Viewer Tool.

3.2. Dynamic Experiments

For the dynamic experiments, data from the movement of the ball in the x, y and z axes was collected and analysed. Because the trajectory of a basketball shot is away and down from the camera, the z axis and the XY plane (vertical plane) were the two subsequent experiments conducted as these outline the trajectory that the ball would take (**Figure 3.8**).

For depth accuracy experiments, two isolated regions were created at a certain distance from the camera to analyse the velocity of the ball (**Figure 3.8** (a)). Each region was designed to capture the centroid coordinates of the ball and the elapsed time between these two centroids from which the velocity at P_1 could be calculated. The following points discuss the details of these specific regions:

- The two separate regions (Region 1 and Region 2 in **Figure 3.8** (a)) were chosen at distances within 2m from the camera as this region was the ideal region to capture the most accurate results.
- Each region looked at z values within a 20cm range to ensure that enough depth values were captured while the ball was moving away from the camera.
- The distance between these two regions was chosen to eliminate the possibility of overlapping depth values from one region into another and test the accuracy in capturing coordinates in the lower and upper bounds of the ideal capturing region.

As a result, the following regions were chosen for both the indoor and outdoor experiments to test how the D455 fared in various lighting conditions:

1. Region 1: 0.6m – 0.8m
2. Region 2: 1.6m-1.8m

A similar approach was taken for the XY plane experiment, where two regions were chosen at a specific distance from the camera (**Figure 3.8** (b)). The movement in vertical and horizontal displacements (XY plane) for the ball in the final experiment will be significantly less than movement of the ball in the z axis. Therefore, only 1m from the camera was the distance chosen for this experiment as this distance produced the most accurate depth data results. Furthermore, this experiment was only conducted indoors.

In both the depth (*z* axis) and XY plane tests:

- Standard deviation was used as a filtering technique and compared with non-filtering results to see how it affects the depth accuracy and velocity accuracy.
- A sample set of 100 3D coordinates were collected at each region from which the centroid coordinates were calculated. This sample size was chosen as it was all the valid coordinates that could be collected within each timeframe, while collecting the closest 3D surface points of the ball.
- A centroid iteration of size 200 was used to calculate the average filtered (using 3 standard deviations) and unfiltered centroids.
- The time was captured at its corresponding captured centroid location using the following method:
    - Start the timer when the program starts running
    - End the timer when the program captures the final centroid coordinate (P_1)
    - The time taken was the elapsed time between capturing the initial centroid (P_0) and capturing the final centroid (P_1).

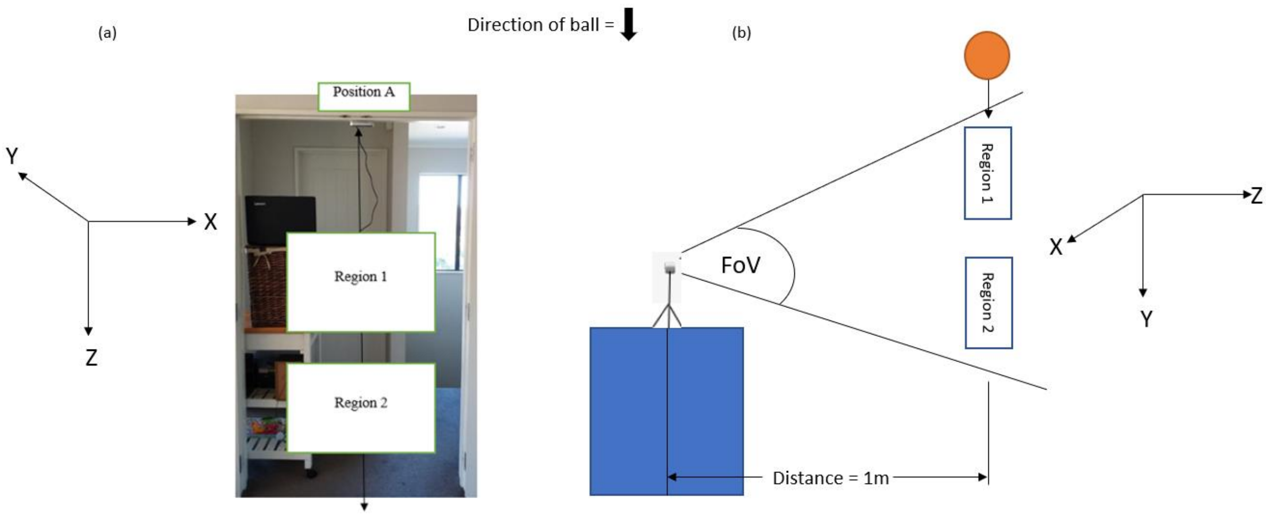

Figure 3.8. Indoor and outdoor dynamic experiment layout for (a) Z Axis (b) Y axis (XY plane) (both not drawn to scale)

3.2.1. Z Axis Experiments

In **Figure 3.8** (a):

- Mount the RealSense D455 facing downwards.
- Run the C++ script in Visual Studio, which will get the 3D centroid coordinates at two separate regions and calculate the velocity of the ball at region 2 (V_1).

- Hold the ball slightly above and in front of the camera (illustrated as position A) and drop the ball. The height between the camera and the ground was approximately 1.96m.
- Capture the centroid coordinates in two separate regions:
  i. Region 1: 0.6m – 0.8m
  ii. Region 2: 1.6m – 1.8m
- Calculate the velocity of the ball at P_1 based on the centroid outputs and the time elapsed between the two regions and then compare the experimental data with the calculated data.
- Repeat the experiment ten times to quantify the variability and get the average velocity.

Figure 3.9 illustrates how the 3D surface coordinates of the ball were captured in mid-air (P_0 and P_1) in the *z* axis.

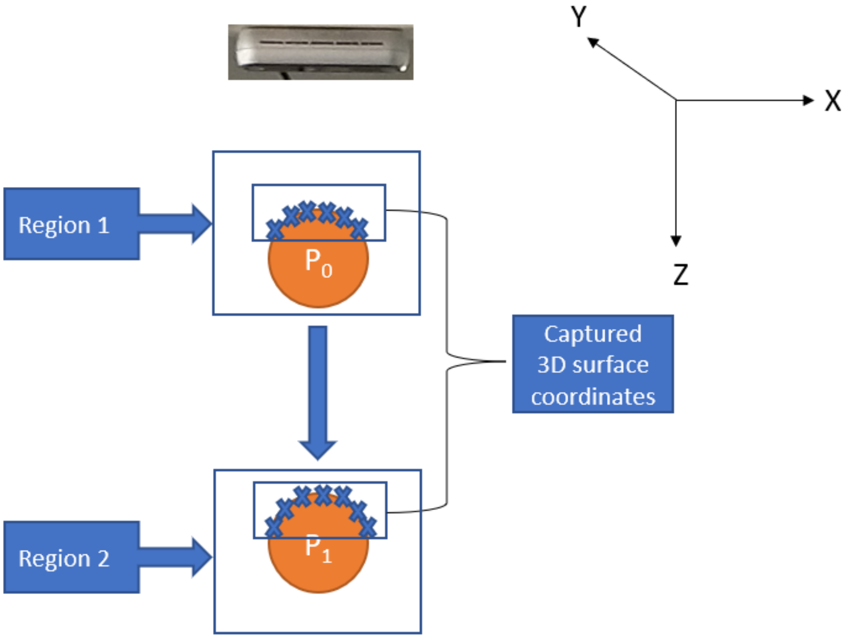

Figure 3.9. The captured 3D surface coordinates of a ball in 2 separate regions moving in the z axis

Each set of 3D surface coordinates captured (100) within these two regions was used to calculate the centroid coordinates (P_0 and P_1) using the same method used in the static experiments. However, the main difference in this experiment was to capture the time each centroid was calculated. **Figure 3.10** illustrates the method outlined below:

1. Two isolated regions were created
2. At each region, a sample set of the first 100 3D coordinates were collected at given a timestamp (t_0 and t_1).
3. From this sample set, the centroid was calculated by taking a set of 4 random coordinates (1 centroid iteration). This process was repeated 200 times to get the average centroid over 200 centroid iterations, each iteration used a random combination of values from the sample set.

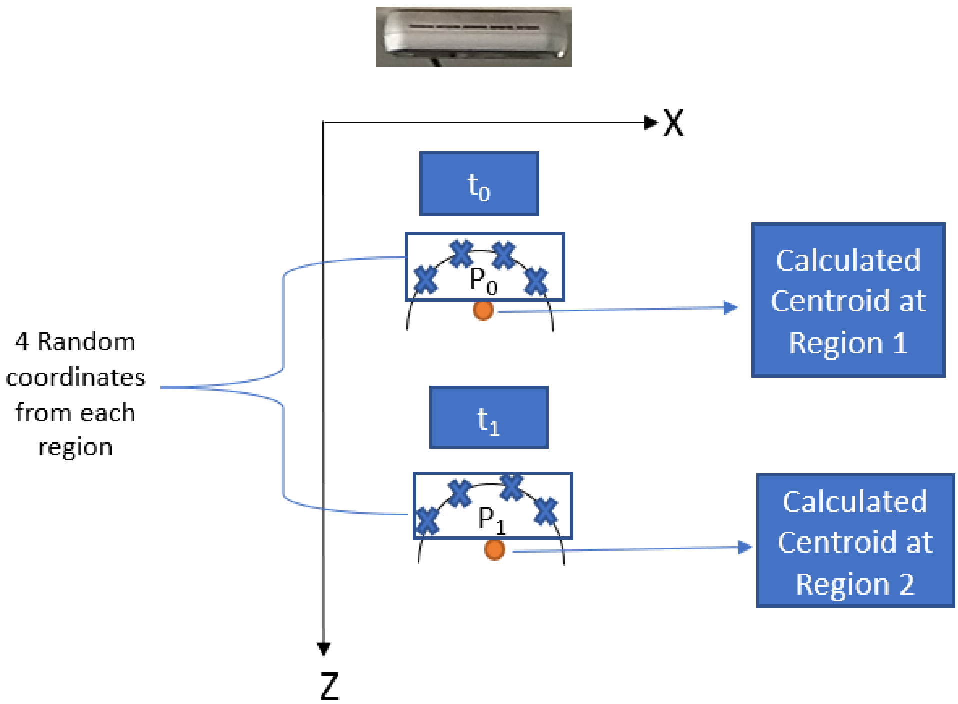

Figure 3.10. Graphic representation of how the centroid was calculated at each region

t_0 and t_1 was the time elapsed from the start of the program to capturing the first 3D coordinate in its corresponding region. From these two centroids, the experimental velocity of the ball was calculated by using the change in displacement of the ball over time (assuming no drag). In the following equations, V_0 refers to the velocity of the ball at the first captured position (P_0) and V_i refers to the initial velocity of the ball. Equations **(5)** and **(6)** are used to calculate the experimental velocity at P_1.

$$(P_1 - P_0) = V_1 \Delta t \tag{5}$$

Manipulating **(5)** to calculate V_1,

$$V_1 = \frac{(P_1 - P_0)}{\Delta t} \tag{6}$$

Where:

V_1: Velocity of the ball at P_1

$\Delta t = t_1 - t_0$: elapsed time in seconds for the ball to reach P_1 from P_0

For the theoretical calculations, the equations of a projectile under gravity under the assumption of negligible drag were used. Equation **(7)** was used to calculate the theoretical time the ball would have reached P_1.

$$t_1 = \sqrt{\frac{2 \times P_1}{g}} \tag{7}$$

From which the theoretical velocity at P_1 was calculated using:

$$V_1 = V_i + gt_1 \tag{8}$$

Because the ball is dropped from a stationary position, $V_i = 0$. The equation then simplifies to:

$$V_1 = gt_1 \tag{9}$$

Where:

g = 9.81 m/s^2

The error between the experimental and theoretical velocity was calculated using **(10)**.

$$V_{error} = V_{calculated} - V_{experimental} \tag{10}$$

Figure 3.11 gives a visual representation of the variables used to calculate the theoretical and experimental velocities.

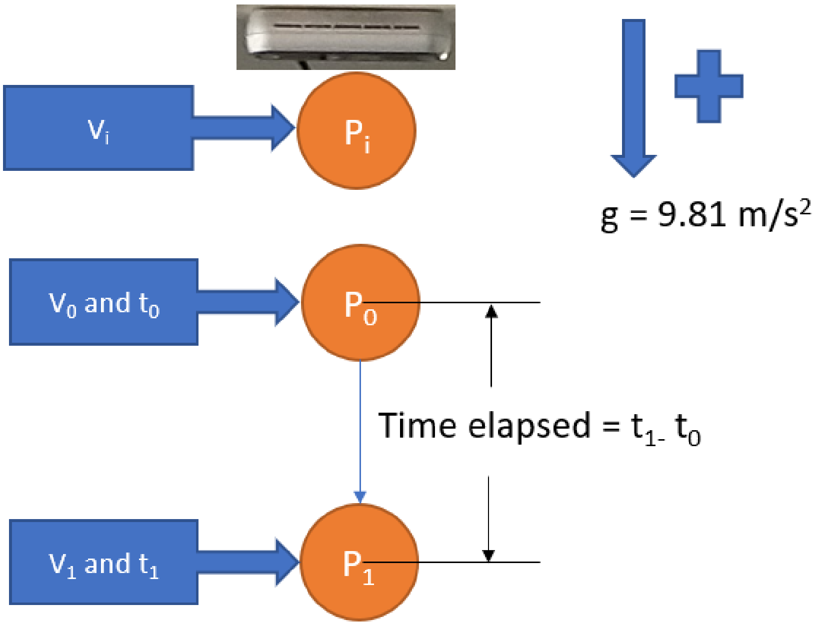

Figure 3.11. A visual representation of the variables used for calculating the velocity in the z axis

In **Figure 3.11**, P_i is the initial position of the ball before it is dropped and P_0 and P_1 are the two calculated centroid positions captured in region 1 and region 2 respectively. These positions, as well as the time elapsed, are captured by the D455 from which the theoretical and experimental velocity at V_1 can be calculated.

3.2.2. XY Plane Experiments

The XY plane experiments are depicted in **Figure 3.8** (b) and illustrated in further detailed in **Figure 3.12** below.

In **Figure 3.12**:

- Stand the RealSense D455 at a 1m distance from the ball.
- Run the C++ script in Visual Studio, which will get the 3D centroid coordinates at two separate regions (Region 1 and Region 2) and calculate the velocity of the ball at P_1.
- Hold the ball slightly above region 1 and drop the ball. The *z* depth range isolated depth values between 1m and 1.5m (illustrated by A).
- Capture the centroid coordinates in two separate regions (**Figure 3.12**):
    i. Region 1: 0.2m height range (B)
    ii. Region 2: 0.2m height range (C)
- Calculate the velocity of the ball at P_1 (using **(5)** and **(6)**) based on the centroid outputs at B and C and the time elapsed between these two regions and then compare the experimental data with the calculated data.
- Repeat the experiment several times to quantify the variability.

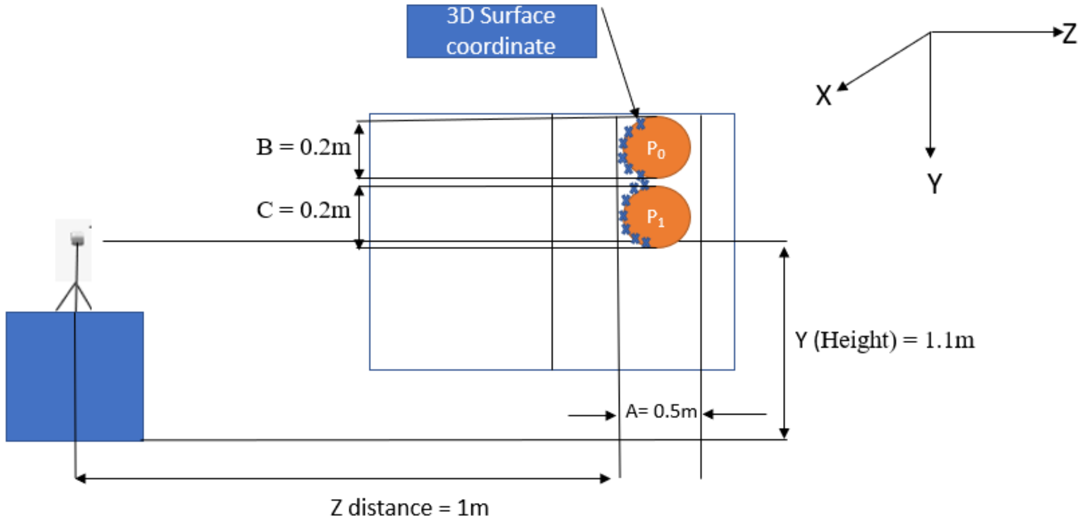

Figure 3.12. Layout for capturing centroid coordinates in the XY plane

In **Figure 3.12**, B and C are the regions in which surface coordinates were captured in the *y* axis and A is the depth ROI that was created to ensure that the camera captured coordinates with different depth values. The same method of calculating the velocity (V_1) was applied.

3.3. Communication between the RealSense and the HoloLens

The UDP protocol was chosen to transfer position data between the D455 and the HoloLens due to its simplicity and high speed. A PC was used as a medium to transfer and process data between the D455 and the HoloLens. Within this experiment, the HoloLens works as the client and the laptop as the server (**Figure 3.13**).

A server-client communication is established through the ending and receiving of IP addresses of both the client and the server. Firstly, the client uses the IP address of the HoloLens to send an initial message to the server (C++). Consequently, the server receives the client's IP address which allows communication between the client and the server. For the 2D trajectory tests, the Intel RealSense SDK is used to calculate the centroid coordinates (P_0 and P_1), which is sent to Unity where the velocity of a ball is calculated, and the resulting trajectory is simulated. This data will then be sent to the client (HoloLens). The messages were received/sent in different threads on the client side which allowed the client to simultaneously listen to the channel while sending a message. Similarly, the server side invoked an event in Unity in response to the message sent by the client which enabled real-time communication between the D455 and the HoloLens.

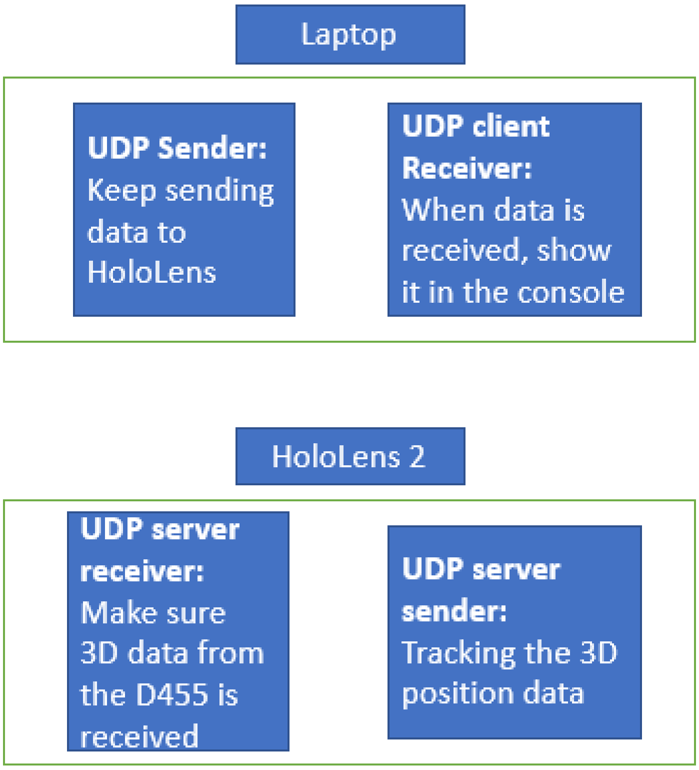

Figure 3.13. Communication architecture between the HoloLens and the D455

Initially, communication was established between the D455 and Unity 3D to test both the static and dynamic experiments and to calibrate the two axes. This was completed as a first step to establish a working connection between the D455 and Unity 3D, which would then serve as the basis to deploy the design onto the HoloLens for final testing of the whole system. The process of sending data via UDP is detailed in the steps below:

1. D455 placed on a flat surface and connected to the PC via USB
2. Run the D455 script on the PC to establish a server connection
3. Establish a server-client communication through a UDP connection
4. Send example centroid coordinates and initial velocity to Unity 3D

This experiment required small sets of 3D data to be sent, therefore UDP was chosen as a form of communication as it is a fast and simple way of sending data. However, the disadvantage of using this form of communication is that it is an unreliable form of communication and data can get lost or not sent through correctly. This is mostly a concern when sending through large data sets. In this case the benefit of speed and simplicity outweighed the need for greater reliability.

When calibrating the two axes, the static 3D position data was sent through to Unity. This required sending four separate 3D centroid coordinates at four separate locations. Once all the centroid positions from the four locations were captured and the two axes were calibrated, the system waited ten seconds before listening for more incoming data. Because the UDP communication constantly listens to incoming messages from the server, the delay between collecting static coordinates and dynamic coordinates was established. Furthermore, this enabled the user to get out of the FoV of the D455 to ensure that no unnecessary data was collected when transitioning to the dynamic experiment and to get into position behind the D455 to conduct the dynamic experiment. Like the static experiment, the centroid coordinates were sent through to the HoloLens. The major difference between the two experiments, was only collecting 3D data at two separate regions and including the initial velocity of the ball in motion.

3.3.1. Communication between Server and Unity3D

The steps for how the communication was established on both ends (server and client) is detailed below:

In C++:

- Initialise the winsock variables
- Initialise the winsock connection
- Create a socket connection between the server and the client
- Prepare the socket address structure, which structures the 3D data that is sent so that the correct 3D data is sent and received
- Bind the server to the client. This function will confirm whether a connection between the server and the client has been established.
- Listen to incoming data from the client and clear the buffer by filling in null as it might have received previous data.
- Try to receive some data (static experiment – centroids, dynamic experiments – centroids and initial velocity)
- Reply to the client with the same data to confirm that the correct data was sent.

In Unity3D:

- Initialise a UDP client object from the System Sockets class
- Initialise 3D vector variables that will read the incoming 3D position data and float variables that will read the time sent at the two locations (from the dynamic experiment)
- Initialise variables for the IP address, receiving port and the sending port
- In the Start() function, establish a connection with the server by sending an initial message to the server.
- Initialise the UDP connection through establishing the correct receiving and sending ports
- Once connection is established, 3D data is sent to the server (C++) in bytes.

- Thereafter, the client listens to incoming 3D data from the server and reads them in readable bytes. Each set of 3D coordinates received, is collected in byte sizes of 8 and then converts the 3D data into readable numerical values.
- It will first collect all the static centroid coordinates, one 3D coordinate per location and calibrate the two axes.
- Then the launch 3D coordinates and the time at two separate locations will be collected, which will be used to calculate the initial velocity of the ball in motion. Both the static and dynamic data will be read the same way (readable bytes).
- Once the dynamic coordinates and the time at the two separate locations have been received, the velocity will be calculated and used as previously described for the ball spawning experiments, which will be discussed in greater detail in section 3.5.

3.3.2. Communication between Unity 3D and HoloLens

Once a working UDP connection from the server to the client was established, the reliability of that data was analysed based on the following two tests:

1. Send 3D coordinates from the server to the client (C++ to HoloLens)
2. Send 10 different 3D coordinates over a certain time interval

The purpose of the first test is to analyse whether the data is sent through correctly from the server to the client. This will confirm if the HoloLens collects the correct 3D coordinate during calibration. The second test will test how reliable the UDP communication is when sending multiple sets of 3D coordinates over a prolonged time and will determine the effectiveness of taking multiple launch positions for the trajectory experiment.

In the first test, two sets of different sized 3D coordinates with random values were sent:

- Set 1 – 5x 3D coordinates
- Set 2 – 10x 3D coordinates

From each set, the sent data from the server side was compared with the received data on the client side and the results were analysed. The second test sent a float value then delayed for six seconds before sending the second float value and was repeated for ten different values. This experiment was designed to best mimic the launch tests conducted in the final experiment and to test how effective the UDP communication would be after sending and receiving multiple values over time.

3.4. Calibration of the RealSense and the HoloLens axes

For this experiment to be conducted, there needs to be a communication established between the D455 and the HoloLens. This was done using UDP communication between these two devices and is outlined in section 3.3. The main idea of the calibration procedure is to get the point coordinates of the same points in different coordinate systems (HoloLens and D455), and then use these corresponding coordinates to calculate the transformation from one coordinate system to the other (D455 to HoloLens in this work). Once the calibration is complete, any 3D points collected by the D455 can be transformed into the coordinate system of the HoloLens. Hence, accurate calibration information allows reprojection of data to a common reference system. Calibration is only needed once, unless the socket connection between the two devices is lost, in that case the calibration process will need to be restarted.

Equipment:

1. HoloLens
2. RealSense D455
3. PC – D455 connected via USB
4. PC – HoloLens connected wirelessly

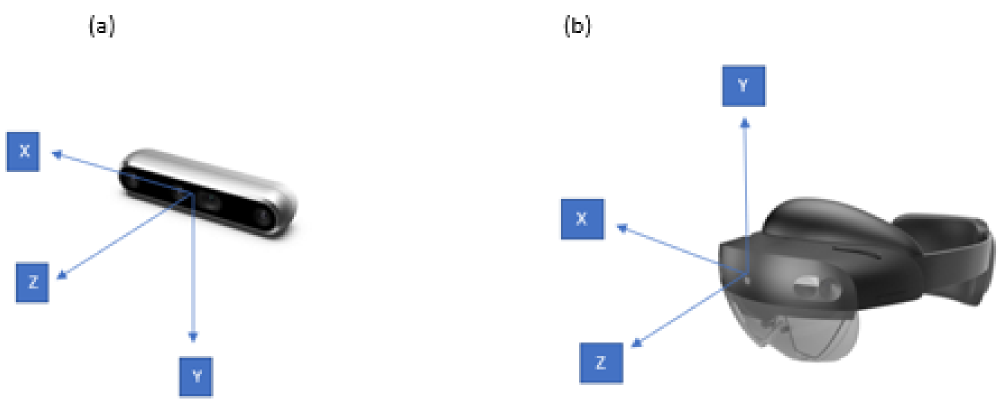

Figure 3.14. (a) Reference frame for the Intel RealSense D455 (Intel) (b) Reference frame for the HoloLens (Bryght, 2022)

For the transformation equations, the rotation and translation components need to be considered. According to the homogeneous transformation, the equation that transforms the coordinates from the D455 reference frame into the reference frame of the HoloLens is:

$$H^A = R_B^A \cdot H^B + T_B^A \quad \textbf{(11)}$$

From ***(11)***, the homogeneous transformation matrix is as follows:

$$\begin{bmatrix} H^A \\ 1 \end{bmatrix} = \begin{bmatrix} R_B^A & T_B^A \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} H^B \\ 1 \end{bmatrix} \quad \textbf{(12)}$$

Where:

A – Denotes System A reference frame

B – Denotes System B reference frame

R_B^A – Denotes rotation of System B with respect to System A

R – Rotation Matrix

T – Translation Matrix

For example, if the *y* axis is rotated 180 degrees about the *x* axis and the *z* axis remains the same then the output would be **(13)**.

$$\left(R_B^A\right)_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\theta & -sin\theta \\ 0 & sin\theta & cos\theta \end{bmatrix}$$

$$\therefore \left(R_B^A\right)_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(13)}$$

The 3D point *H* in system *B* is denoted as follows:

$$H^B = \begin{bmatrix} H_x^B \\ H_y^B \\ H_z^B \end{bmatrix} \quad \textbf{(14)}$$

The D455 is translated by a certain distance (unknown in this example) in the *x*, *y* and *z* axes.

$$T_B^A = \begin{bmatrix} (T_B^A)_x \\ (T_B^A)_y \\ (T_B^A)_z \end{bmatrix} \quad \textbf{(15)}$$

Using **(12)**, the completed homogeneous transformation matrix is seen below:

$$\therefore H^A = \begin{bmatrix} 1 & 0 & 0 & (T_B^A)_x \\ 0 & -1 & 0 & (T_B^A)_y \\ 0 & 0 & 1 & (T_B^A)_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} H_x^B \\ H_y^B \\ H_z^B \\ 1 \end{bmatrix}$$

$$\therefore H^A = \begin{bmatrix} H_x^B + (T_B^A)_x \\ -H_y^B + (T_B^A)_y \\ H_z^B + (T_B^A)_x \\ 1 \end{bmatrix} \qquad \textbf{(16)}$$

3.4.1. ICP Algorithm in C++

The calibration of the two axes was performed using the Iterative Closest Point (ICP) algorithm, also known as a point set (cloud) registration algorithm. This technique is widely used in the field of computer vision, pattern recognition, robotics, and image processing, hence why it was chosen for this experiment. The main purpose of point set registration is to find correspondences and estimate the transformation (rotation and transformation) between two or more point-sets. However, in this case the correspondences are already known as we have a fixed set of known coordinates from the HoloLens. These coordinates will be compared with the corresponding D455 coordinates, which will be collected at four different positions. Both sets of coordinates will be used to calculate an appropriate transformation matrix. Before doing calibration of these two devices, some important factors were considered, namely:

- Field of View (FoV) of each camera (horizontal and vertical)
- Resolutions and nature of imaging system of each camera
- Colour of each camera
- Infrared (IR)
- Depth of each camera
- Extrinsic and intrinsic parameters of each camera
- Spatial coordinate (3D) system for each camera to determine the relationship between the 2 coordinate systems.

In this experiment, the D455 will be referred to as the data shape and the HoloLens referred to as the model shape. Using the ICP method, 3-D data point cloud is used to describe how the data shape corresponds to the model shape. The aim of this algorithm is to estimate the optimal rotation and translation that aligns the model shape and the data shape minimizing the distance between the shapes via a mean-square distance metric.

The first step of the algorithm is to compute the closest point to a given point, also known as obtaining correspondences. However, in this experiment, the HoloLens points are already known and are the only corresponding closest points that need to be considered. The next step is to calculate the rigid transformations - the rotation and translation matrices, known as the point set registration. This will be completed using Single Value Decomposition (SVD) as described by (Rajkumar, 2021) in Least Square Fitting of Two 3-D Point.

Given two separate point sets *P* and P^I are related by:

$$P^I = RP + T + N \tag{17}$$

Where, *R*: Rotation matrix, *T*: Translation Matrix and *N* is a noise vector, then this algorithm presents a way of finding the least-squares solution of *R* and *T* based on SVD of a 3x3 matrix.

To find *R* and *T*, minimise

$$\Sigma^2 = \sum\nolimits_{n=1}^{N} |P^I - (RP + T)|^2 \tag{18}$$

This is completed by firstly finding the centroid of both the point clouds by finding their respective means. Thereafter, we centre the mass-point of each point cloud by subtracting the centroid values from their respective point clouds.

If *Q* and Q^I are the centroid mass-points of *P* and P^I respectively we have:

$$\Sigma^2 = \sum\nolimits_{n=1}^{N} |Q^I - RQ|^2 \tag{19}$$

This reduces the least-squares problem to two parts:

1. Find *R* by minimizing the above equation
2. Find translation, *T* using:

$$T = P^I + R_P \tag{20}$$

Once *P*, P^I, *Q* and Q^I are obtained, a 3x3 covariant matrix can be calculated as follows:

$$H = \sum_{i=1}^{\infty} Q_i Q_i^t \tag{21}$$

This code implementation is referred to in (Appendix A, **Figure A 3**).

The next step was to find the SVD of H. C++ uses functions that help in calculating the SVD of H which gives matrices U and V^t. Thereafter, the rotation matrix, R is calculated as follows:

$$R = VU^t \tag{22}$$

From the R matrix above, the determinant is calculated and if the det(R) is 1, then the rotation matrix was calculated correctly, and the calculation of T can be calculated from **(22)**. However, if the det(R) is -1, the algorithm fails and leads to a reflection instead of rotation. This issue can be fixed by multiplying the last column of the V matrix with -1, but only works if two of the three singular values of H are equal. This transformation code is referred to in (Appendix A, **Figure A 4**).

The final step of the ICP algorithm is aligning the two point clouds. The data shape point cloud P is moved to best align with the shape point cloud M. The following steps are repeated until convergence:

- Compute registration using SVD
- Apply computed registration to the data point cloud
- Terminate the process when the mean-square error falls below a pre-set threshold or if the total number of iterations have been exhausted.

During the testing of this algorithm, with random example values, the correct solution of the transformation matrix was calculated after a single iteration (Appendix A, **Figure A 5**). This eliminated the need to find the mean-square distance between the desired model shape and the data shape over multiple iterations. The reason for this could be due to a small data set, which means that it reaches the optimal solution much faster.

3.4.2. Design Layout for ICP Calibration in Unity

The ball calibration procedure within Unity was conducted at four fixed locations. These coordinates were used as the model shape. Thereafter, the D455 would capture the centroid coordinates of the ball at each of those four coordinates to calculate the transformation matrix, which will align both coordinate axes.

The following steps outline the calibration procedure within the Unity environment:

1. Within the AR scene on the HoloLens, position four spheres that are slightly bigger than the size of the basketball. All these spheres will have known 3D centroid coordinates.
2. Place AR sphere at four different locations within the room and within the FoV of both sensors. The known coordinate values (in m) for each ball are outlined below and illustrated in **Figure 3.15**:

a. Ball 1: (0,0,0.75)
b. Ball 2: (0.05,0.2,1)
c. Ball 3: (0.1,0.4,1.25)
d. Ball 4: (0.15,0.6,1.5)

3. Before each centroid coordinate of the ball is captured by the D455, the spherical ball on the HoloLens was coloured white. Once the coordinate has been captured, the ball turned green, indicating that the system has captured the 3D coordinate corresponding to that specific location. This method used the transformation matrix determined from the calibration to transform known location(s) according to the following:
    a. Calculate the position of each calibration ball (known from the perspective of the HoloLens), do this 10 times for each ball
    b. Calculate the difference between the calculated values and the known value for each ball and then report the mean and standard deviations of these which will show the error (accuracy and precision) in the calibration
4. Step 3 will be repeated until all four centroid coordinates have been captured.
5. Cameras and calibration target remain fixed during calibration to avoid misalignment and motion blur.

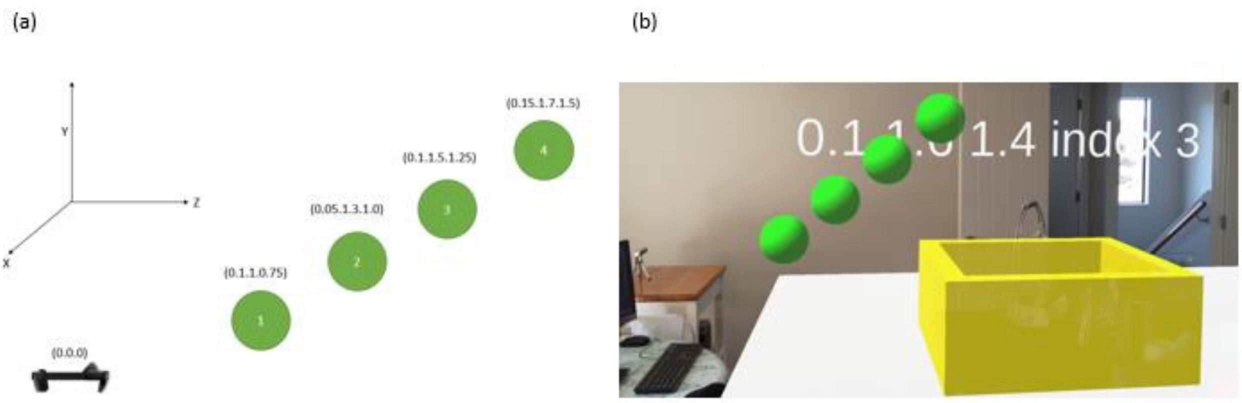

Figure 3.15. Four calibrated spheres at the four different locations (a) design (not drawn to scale) (b) view from HoloLens

Figure 3.15 shows the green calibrated spheres after calibration is completed in terms of its design and from the view of the HoloLens. Once all four coordinates have been obtained, the ICP algorithm will calculate the transformation matrices (R and T), which will be used to transform the coordinates obtained during dynamic testing.

The ICP is mostly used to calibrate point clouds with large data sets as it works well in reducing the noise parameter from outlying data points. However, the purpose for using four calibration points is to make calibration testing convenient to use by reducing the number of centroid calibration points. Therefore, there is a higher probability of obtaining noisy data, which will affect the overall accuracy of the 2D trajectory experiment discussed below.

3.5. 2D Trajectory accuracy in HoloLens

The 2D trajectory experiments use the D455 to send 3D position data to the HoloLens using UDP communication. Before the 3D position data is sent through to the HoloLens, a connection is established between the server and client (discussed in section 3.3). Thereafter, the two coordinate axes (D455 and HoloLens) are calibrated using the ICP method (discussed in 3.4.1). Once calibration has been completed, the HoloLens can receive the 3D coordinate data in real-time. This is performed by throwing a ball towards a target at a specific distance from the camera. The purpose of this experiment is to design a system that can hit a desired target area at a certain distance from the camera through simulating the trajectory of the ball. **Figure 3.16** illustrates how the D455 will simulate the trajectory of the ball at P_1 based on P_0.

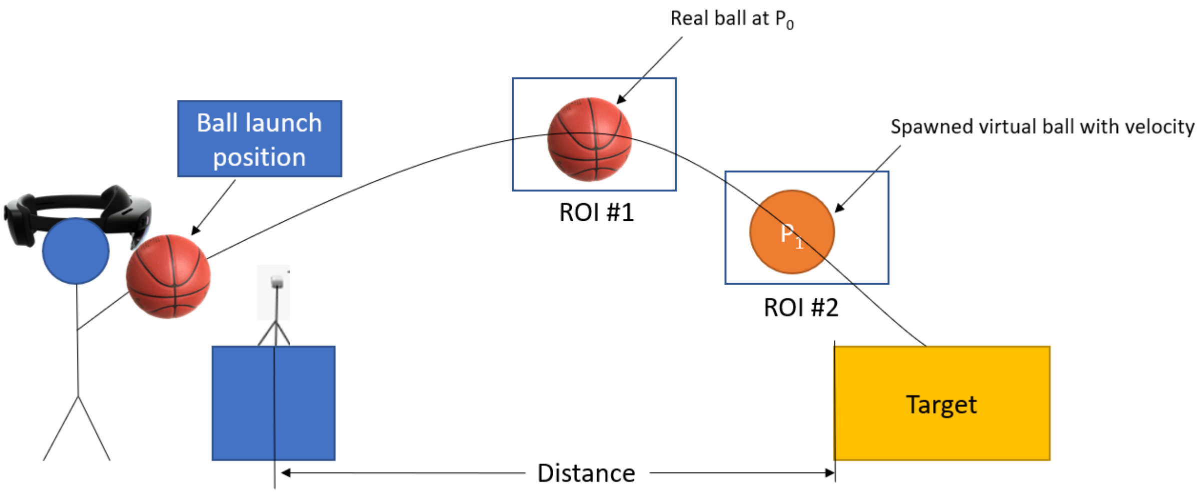

Figure 3.16. Overall setup of the ball's 2D trajectory simulation experiment using the D455 and the HoloLens

In **Figure 3.16**, the capturing regions (denoted by ROI) for each box (denoted by target) was adjusted to suit the distances at which each box was placed. The distances and capturing regions for each box are detailed in the experimental method below.

3.5.1. Experimental Outline

1. Create the AR environment inside Unity, including a virtual ball with rigid body properties (gravity and elasticity) that properly simulated forces acting (i.e., drag free projectile motion as deemed appropriate in the literature review) on a real ball in mid-air and create a virtual target at three different locations from the D455.

Within these distances, three target areas of 0.5mx0.5m were chosen as easy targets to hit in real life. The distances of these targets from the camera were chosen to be within the ideal depth range (1m-2m) of the D455 to get a smaller error in the estimated launch conditions. Each target was placed at the following positions:

a. Box 1: 0.7m-1.2m
b. Box 2: 1.2m-1.7m
c. Box 3: 1.7m-2.2m

The same method of capturing 2 separate coordinates in two separate regions (mentioned in section 3.2) was used for this experiment. For each box, the capturing regions were adjusted to suit the distances at which each box was placed. These regions are outlined below:

- Box 1:
  - Region 1: 0.6m-0.8m
  - Region 2: 1m-1.2m
- Box 2:
  - Region 1: 0.8m-1m
  - Region 2: 1.2m-1.4m
- Box 3:
  - Region 1: 1m-1.2m
  - Region 2: 1.6m-1.8m

2. Create a Unity script that can receive position coordinates and then simulate the ball's trajectory using its in-built physics engine.
3. Enable the correct settings to deploy the Unity project onto the HoloLens.

The MRKT package was installed in Unity and used to deploy the virtual environment created in Unity onto the HoloLens. UWP was the required platform for deployment of the Unity3D environment onto the HoloLens. Thereafter, the correct build settings were established to suit deployment onto the HoloLens (**Figure 3.17**).

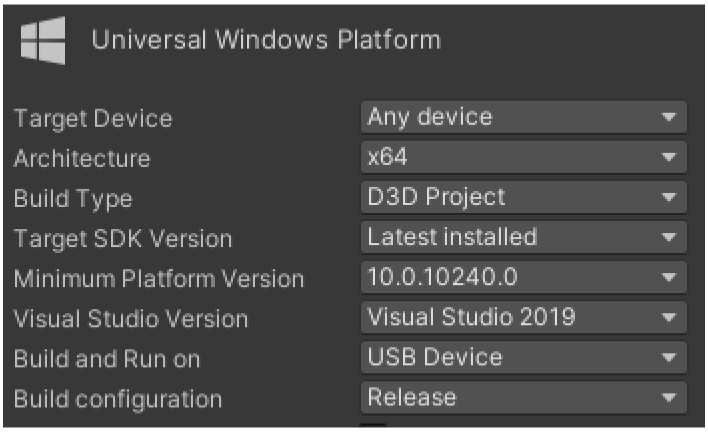

Figure 3.17. Build settings for UWP needed to deploy onto the HoloLens

The Unity3D code is then remotely deployed onto the HoloLens over a Wi-Fi connection using the IPv4 address of the HoloLens in the project properties tab inside Visual Studio as illustrated in **Figure 3.18**.

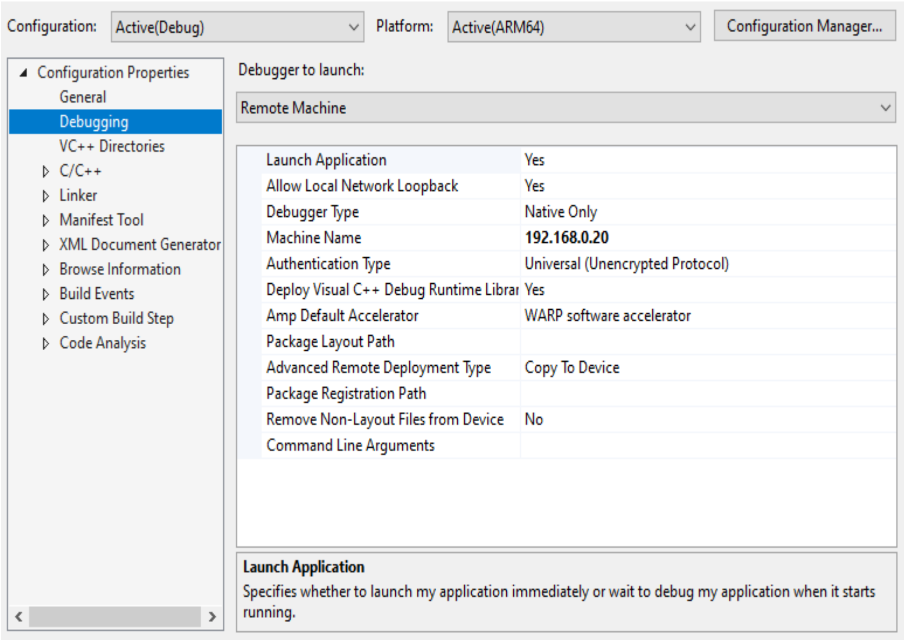

Figure 3.18. Input correct IPv4 address (Machine Name) to establish remote connection

To support deployment onto the HoloLens, inside Visual Studio the following settings were changed:

a. Solutions configuration – Debug
b. Solutions platform - ARM64
c. Run configuration - Remote Machine

Lastly, the HoloLens is turned on and the configuration settings run inside Visual Studio. Once the project has been built and the deployment is successful, the environment will open in the HoloLens and will allow data to be sent via UDP through a datagram socket connection.

4. Place the real target (box) of a similar size at the same location as the virtual targets outlined above.
5. Run the C++ server to enable 3D position data to be sent to the HoloLens (outlined in section 3.3.1).
6. Load the deployed Unity project on the HoloLens and calibrate axes.
7. Launch the ball from behind the D455 and record the number of real balls that hit the target and compare them with the simulated trajectories from the spawned balls.

A total of 30 launches were conducted at each distance to obtain enough data for each test. For each test, a t-test between the number of successful real-world shots versus the number of successful virtual world shots was conducted to determine if the virtual world results could be considered different to the real-world results (a two tailed t-test with a significance level of 0.05 was used). As a final experiment, for both the calibration and launch error, a mathematical prediction of a typical shot trajectory (a free throw) was completed from which the impact of the launch error on the prediction of the ball path was shown.

This was done by showing a theoretically shot free throw at a launch velocity that goes in (ideal prediction path) and then using the largest launch errors (from 3 standard deviations) in the z and y axis to calculate the undershoot and overshoot prediction paths. An example of an ideal free throw launch velocity, launch height and launch angle were 7.47m/s, 2.1m and 53 degrees respectively taken from the free throw line 4.57m away from the release of the ball(Okubo & Hubbard, 2006). The calibration errors were ignored as they had a negligible impact on the prediction path in comparison to the error in velocity. The equations used to calculate the three trajectory paths are outlined below:

<u>For the ideal trajectory:</u>

- z axis

$$z = z_0 + V_z t \tag{23}$$

- y axis

$$y = y_0 + V_y t - 0.5gt^2 \tag{24}$$

<u>For the undershoot and the undershoot trajectories:</u>

- z axis

$$z = z_0 + (V_z \pm e_z)t \tag{25}$$

- y axis

$$y = y_0 + (V_y \pm e_y)t - 0.5gt^2 \tag{26}$$

Where:

- z_0 - Initial horizontal position
- y_0 - Initial vertical position
- V_z – Ideal velocity in the z axis
- V_y – Ideal velocity in the y axis
- e_z – Error velocity in the z axis
- e_y – Error velocity in the y axis
- g = 9.81m/s^2

(23) and **(24)** are used to calculate the *z* and *y* positions of the trajectory with respect to time, whereas ***(25)*** and **(26)** calculate these trajectories with a *z* and *y* launch error.

Chapter 4 Results and Discussion

Based on the methods discussed in Chapter 3, this section will analyse the results from the five major experiments performed namely:

1. Static
2. Dynamic
3. Communication
4. Calibration
5. Simulation of a ball's trajectory

The first two sections detailed the results taken from the D455 alone, whereas the communication and calibration experiments served as important intermediary experiments on which the final experiment relied.

4.1. Static Analyses

For the static results, the accuracy of the *x*, *y* and *z* axes were analysed at four different distances from the D455 by comparing the centroids calculated from the algorithm (experimental data) with the measured data (ground truth data). From these two data sets, each axis at each distance (1m, 2m, 3m and 4m) were analysed. The following factors were important in identifying the accuracy of the captured data:

2. Sample set size
3. Centroid set iterations
4. Unfiltered data vs filtered data (filtering data outside 3 standard deviations)

Two sample set sizes of 200 and 1000 3D coordinates were analysed from which the centroid was calculated with and without filtering based on standard deviation. The z centroid measurement of the ball was estimated using a tape measure and the Intel RealSense Viewer, whereas the x and y centroids were estimated from the Intel RealSense Viewer only.

4.1.1. At 1m

Centroid distance from camera:

- RealSense = 1m
- Tape Measurement = 0.99m
- Error % = 1%

Figure 4.1 to **Figure 4.4** show the error in the *x*, *y* and *z* axes between the experimental data and ground truth data for two sample sizes (200 and 1000 3D coordinates) at 1m away from the D455 over multiple centroid iterations (200, 400, 600, 800 and 1000). For each sample set, the unfiltered values are compared with the filtered values at 3 standard deviations from which the average error for each centroid iteration set is calculated.

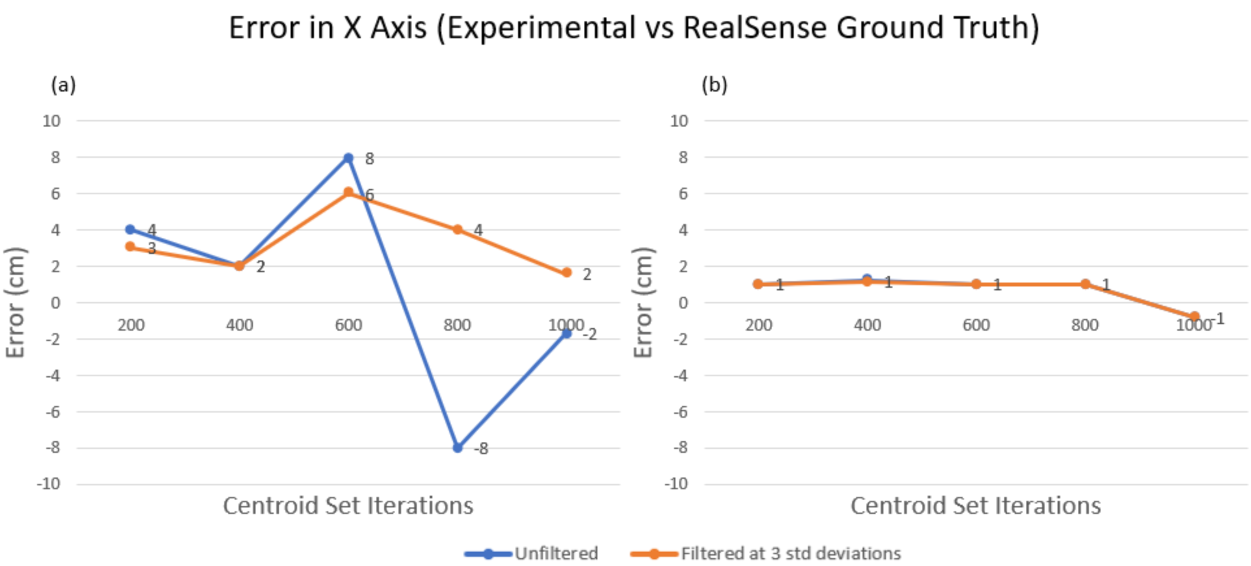

Figure 4.1. A graphical representation of the error in the *x* axis from the unfiltered data vs the filtered data at 3 standard deviations at 1m where (a) Sample size = 200 (b) Sample size = 1000

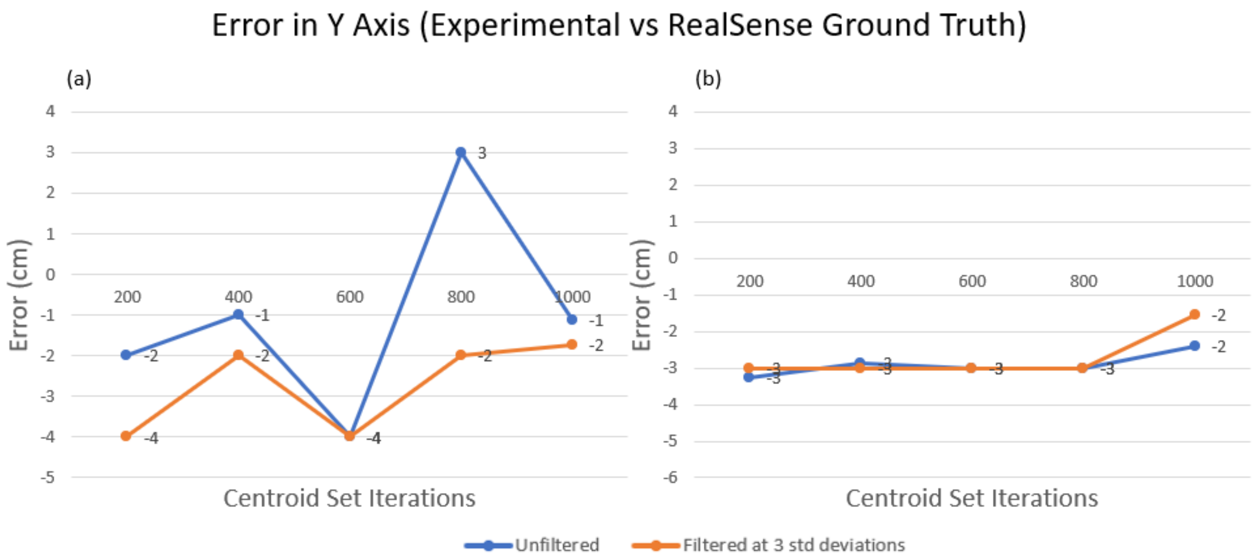

Figure 4.2. A graphical representation of the error in the *y* axis from the unfiltered data vs the filtered data at 3 standard deviations at 1m where (a) Sample size = 200 (b) Sample size = 1000

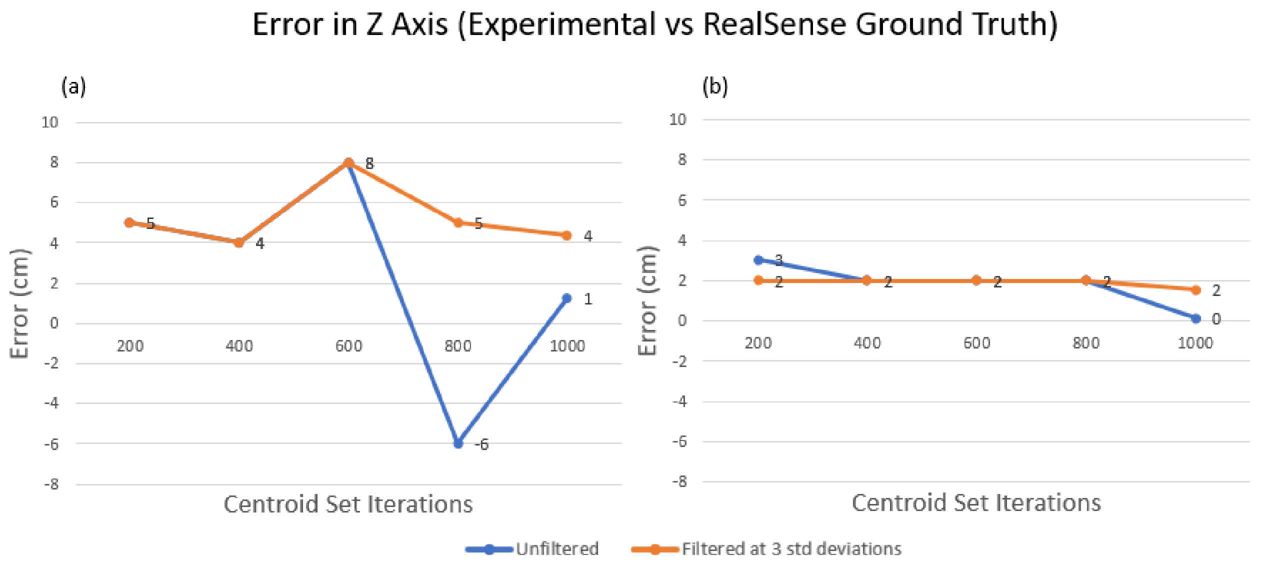

Figure 4.3. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 1m where (a) Sample size = 200 (b) Sample size = 1000

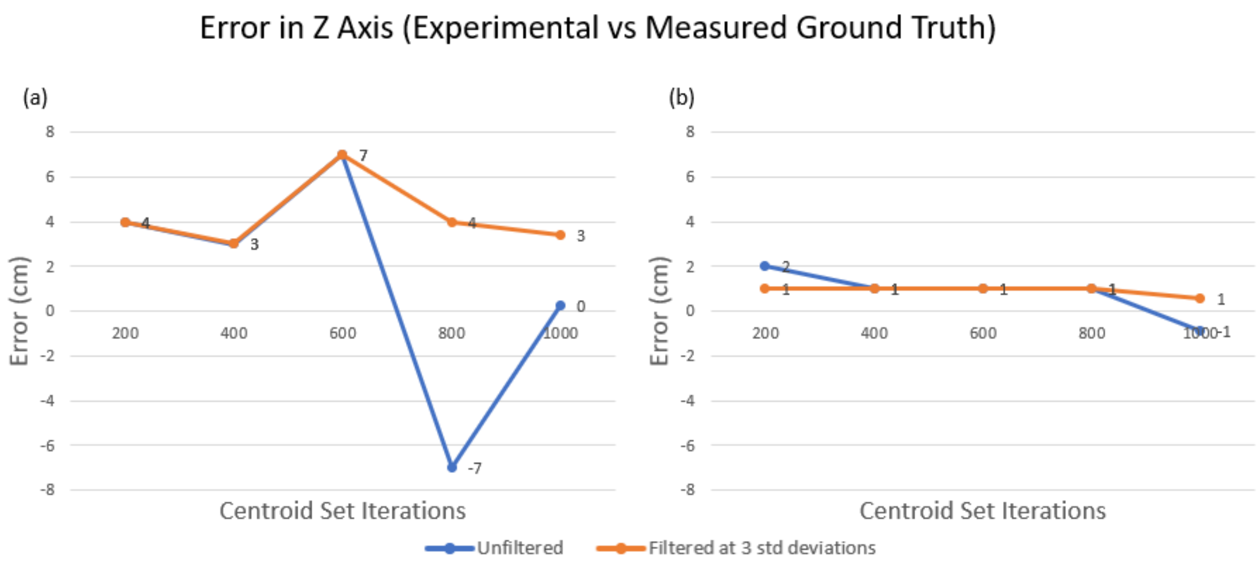

Figure 4.4. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 1m where (a) Sample size = 200 (b) Sample size = 1000

In **Figure 4.1** and **Figure 4.3**, the largest error was 8cm both in the sample set size of 200, whereas sample set of size 1000 showed a largest error of 3cm in **Figure 4.2** and **Figure 4.3**. Overall, the sample set size of 200 showed larger error results compared with the sample set of 1000 and the centroid iterations of 1000 resulted in the most accurate results in both sample sets. The filtering technique occasionally reduced the large unfiltered errors as depicted in **Figure 4.1** and **Figure 4.4**, by 4cm and 3cm, respectively. However, the overall effect of this filtering technique had a negligible effect on reducing the overall error in the results, perhaps due to the greater-known accuracy of the device at close distances. The worst errors of 8cm occurred in the *x* and *z* axes (seen in **Figure 4.1** and **Figure 4.3**). However, there was no single axis that produced the worst average error results.

4.1.2. At 2m

Centroid distance from camera:

- RealSense = 2m
- Tape Measurement = 1.96m
- Error % = 2%

Figure 4.5 to **Figure 4.8** show the error in the *x*, *y* and *z* axes between the experimental data and ground truth data for two sample sizes (200 and 1000 3D coordinates) at 2m away from the D455 over multiple centroid iterations (200, 400, 600, 800 and 1000). At each sample set, the unfiltered values are compared with the filtered values at 3 standard deviations from which the average error for each centroid iteration set is calculated.

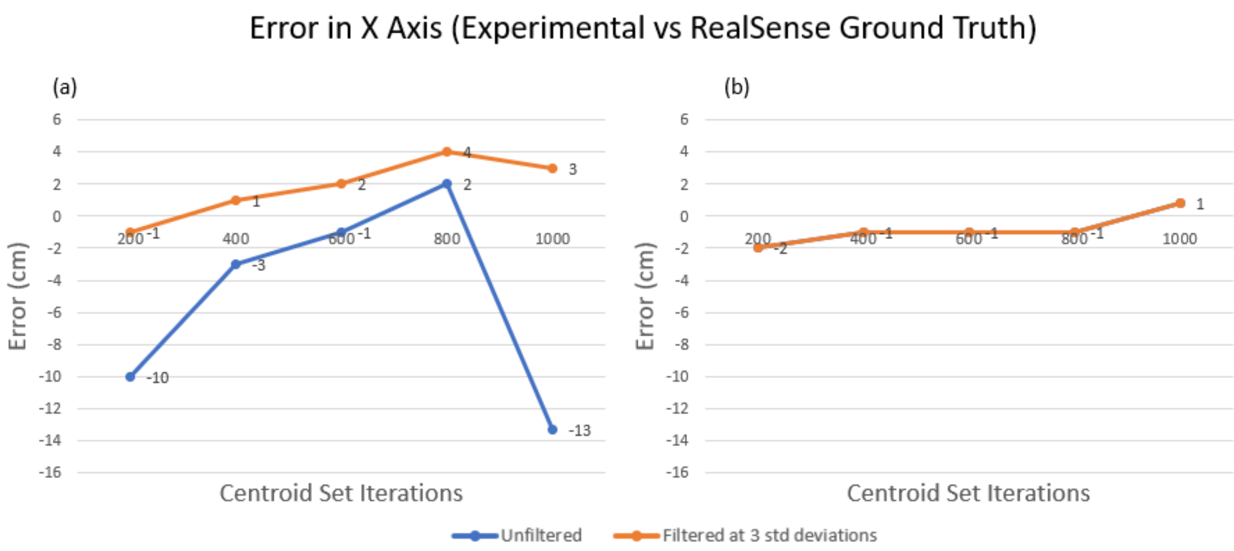

Figure 4.5. A graphical representation of the error in the *x* axis from the unfiltered data vs the filtered data at 3 standard deviations at 2m where (a) Sample size = 200 (b) Sample size = 1000

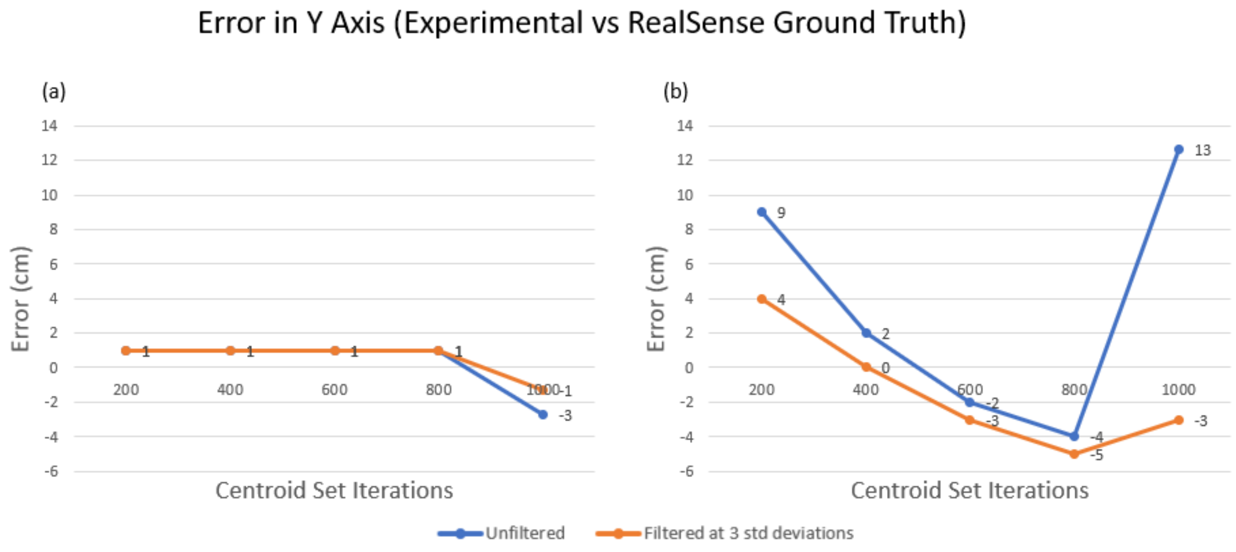

Figure 4.6. A graphical representation of the error in the *y* axis from the unfiltered data vs the filtered data at 3 standard deviations at 2m where (a) Sample size = 200 (b) Sample size = 1000

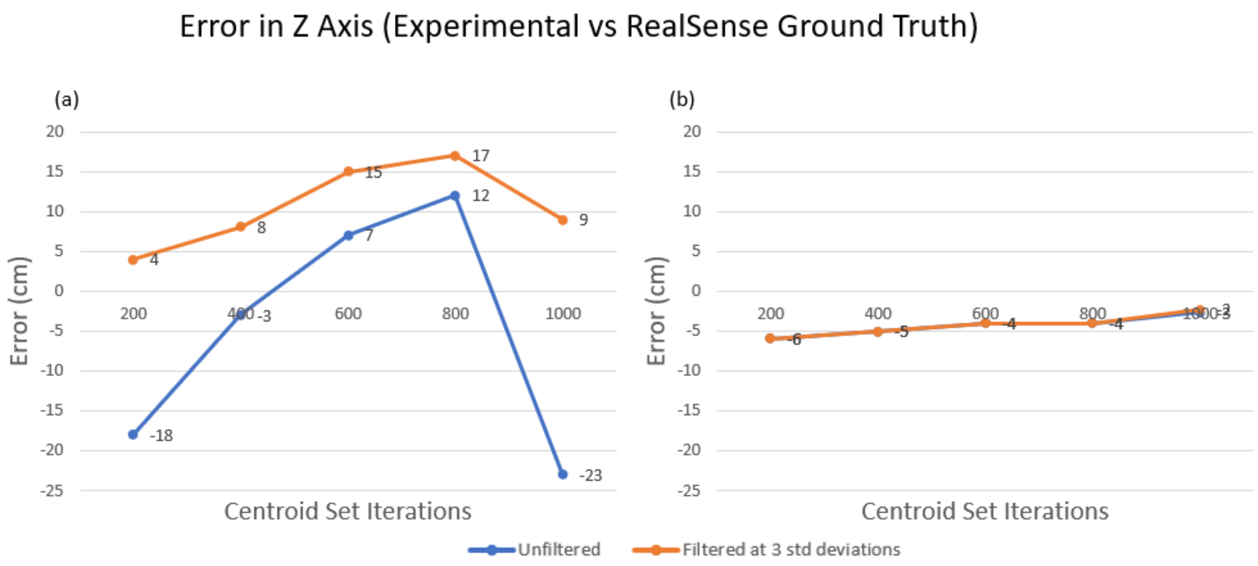

Figure 4.7. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 2m where (a) Sample size = 200 (b) Sample size = 1000

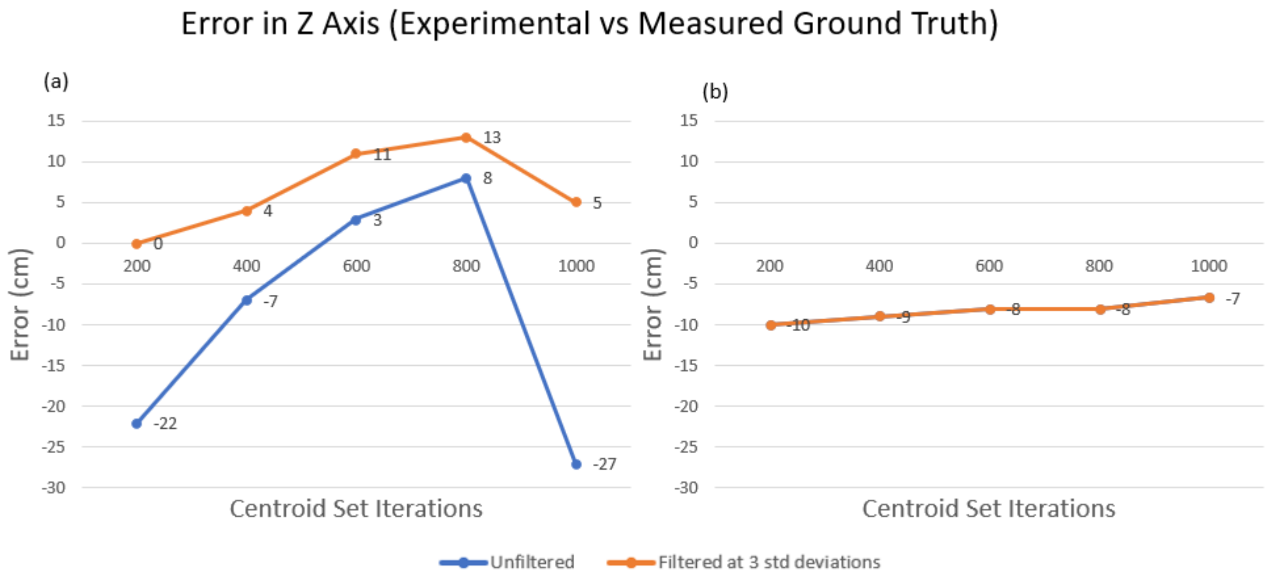

Figure 4.8. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 2m where (a) Sample size = 200 (b) Sample size = 1000

From **Figure 4.8**, the worst error in the data was -27cm in the *z* axis at a sample size of 200 and at the centroid set iteration of 1000. In comparison, the worst error seen in the sample set of 1000 was -10cm in the *z* axis. Moreover, for the sample set of 200, the filtering technique managed to reduce the error by as much as 22cm and managed to reduce multiple large errors overall, whereas the number of centroid set iterations had no effect on reducing the errors from either sample set. Like the results from 1m, the sample size of 1000 proved to produce results with smaller errors compared with sample size of 200. The axis with the highest and more frequent errors was the *z* axis (depicted in **Figure 4.7** and **Figure 4.8**). The worst error from these tables (27cm) was more than double the worst error seen in both other axes (*x* and *y*).

4.1.3. At 3m

Centroid distance from camera:

- RealSense = 3m
- Tape Measurement = 2.88m
- Error % = 4%

Figure 4.9 to **Figure 4.12** show the error in the *x*, *y* and *z* axes between the experimental data and ground truth data for two sample sizes (200 and 1000 3D coordinates) at 3m away from the D455 over multiple centroid iterations (200, 400, 600, 800 and 1000). At each sample set, the unfiltered values are compared with the filtered values at 3 standard deviations from which the average error for each centroid iteration set is calculated.

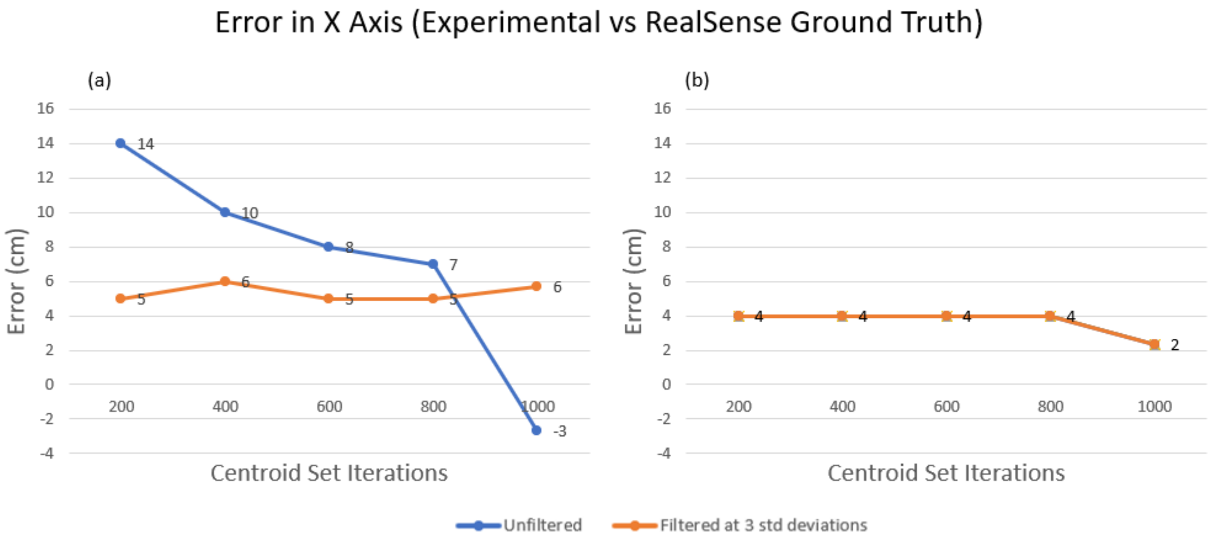

Figure 4.9. A graphical representation of the error in the *x* axis from the unfiltered data vs the filtered data at 3 standard deviations at 3m where (a) Sample size = 200 (b) Sample size = 1000

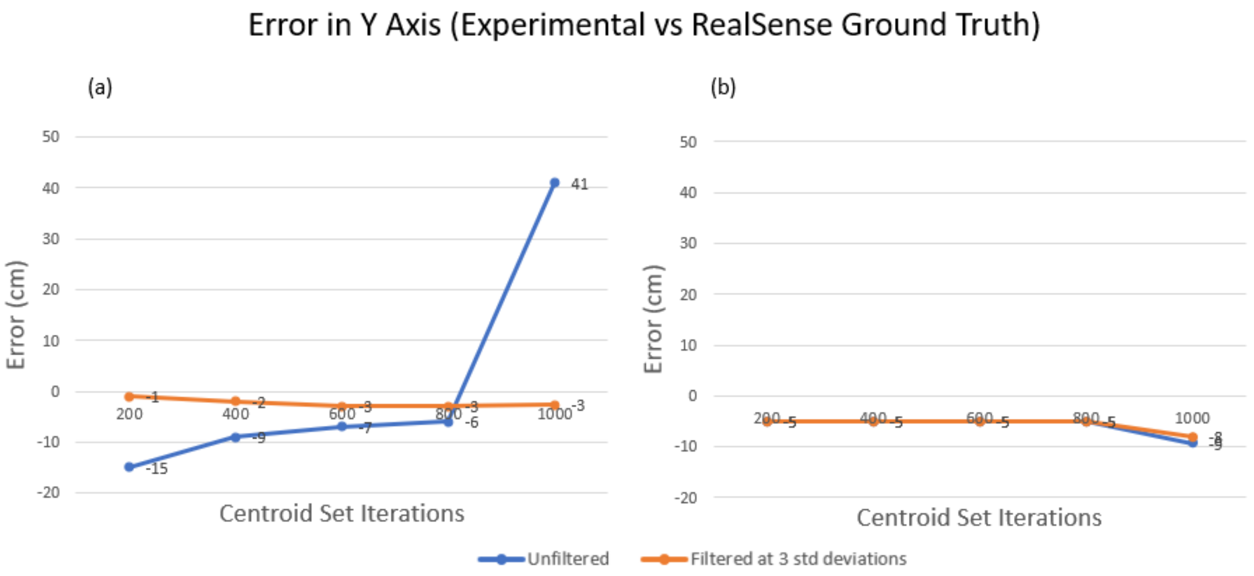

Figure 4.10. A graphical representation of the error in the *y* axis from the unfiltered data vs the filtered data at 3 standard deviations at 3m where (a) Sample size = 200 (b) Sample size = 1000

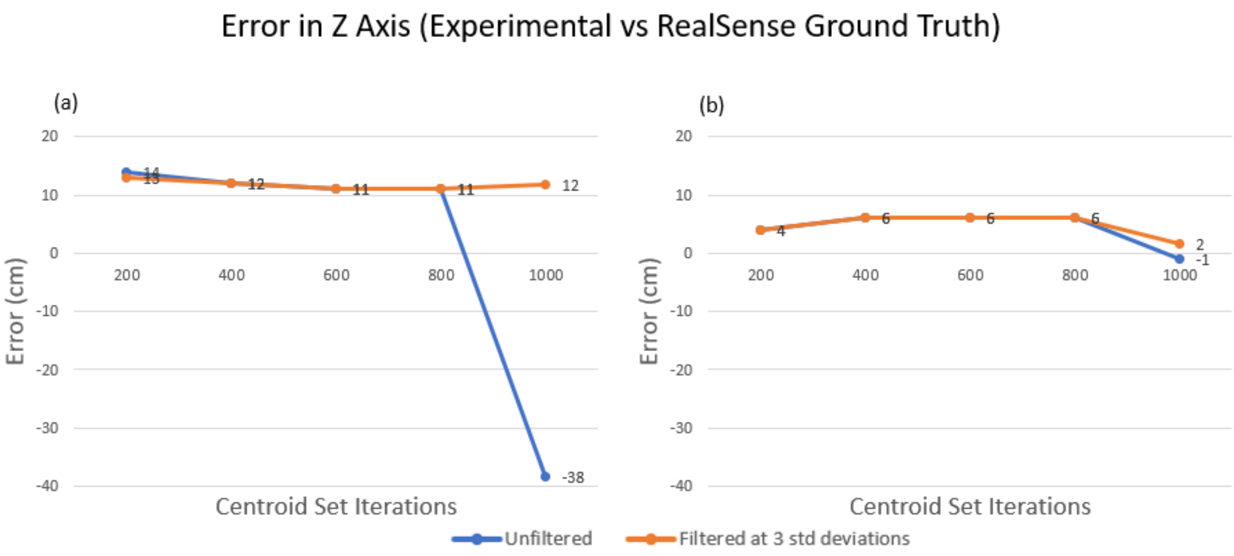

Figure 4.11. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 3m where (a) Sample size = 200 (b) Sample size = 1000

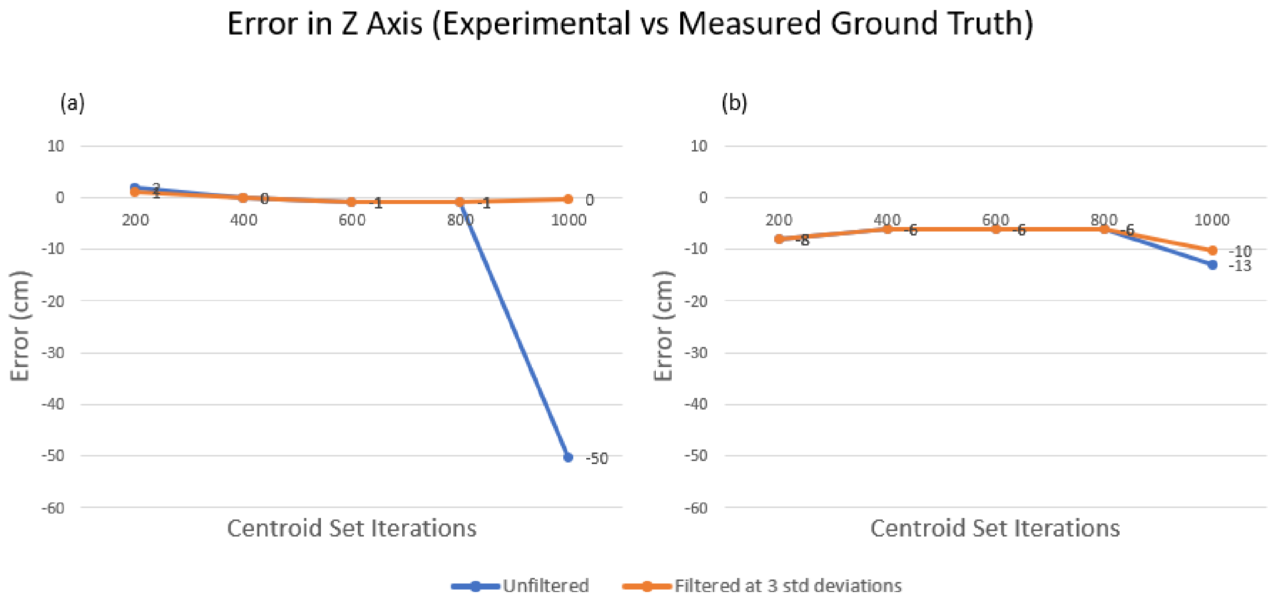

Figure 4.12. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 3m where (a) Sample size = 200 (b) Sample size = 1000

Like the results at 2m, the largest error in the data was in the z axis (-50 cm in **Figure 4.12**) at a sample size of 200 and at the centroid set iteration of 1000. At the sample set of 200 in this figure, the filtering technique managed to reduce the error by as much as 50cm. From this figure, the worst that came from the sample set of 1000 was 13cm in the *z* axis. Overall, the filtering managed to reduce multiple large errors, whereas the centroid iteration size had no effect on reducing the errors from either sample set. Like the results from 1m and 2m, compared with sample size of 200, the sample size of 1000 proved to produce results with smaller errors.

4.1.4. At 4m

Centroid distance from camera:

- RealSense = 4m
- Tape Measure = 3.78m
- Error % = 5.5%

Figure 4.13 to **Figure 4.16** show the error in the x, y and z axes between the experimental data and ground truth data for two sample sizes (200 and 1000 3D coordinates) at 4m away from the D455 over multiple centroid set iterations (200, 400, 600, 800 and 1000). At each sample set, the unfiltered values are compared with the filtered values at 3 standard deviations from which the average error for each centroid iteration set is calculated.

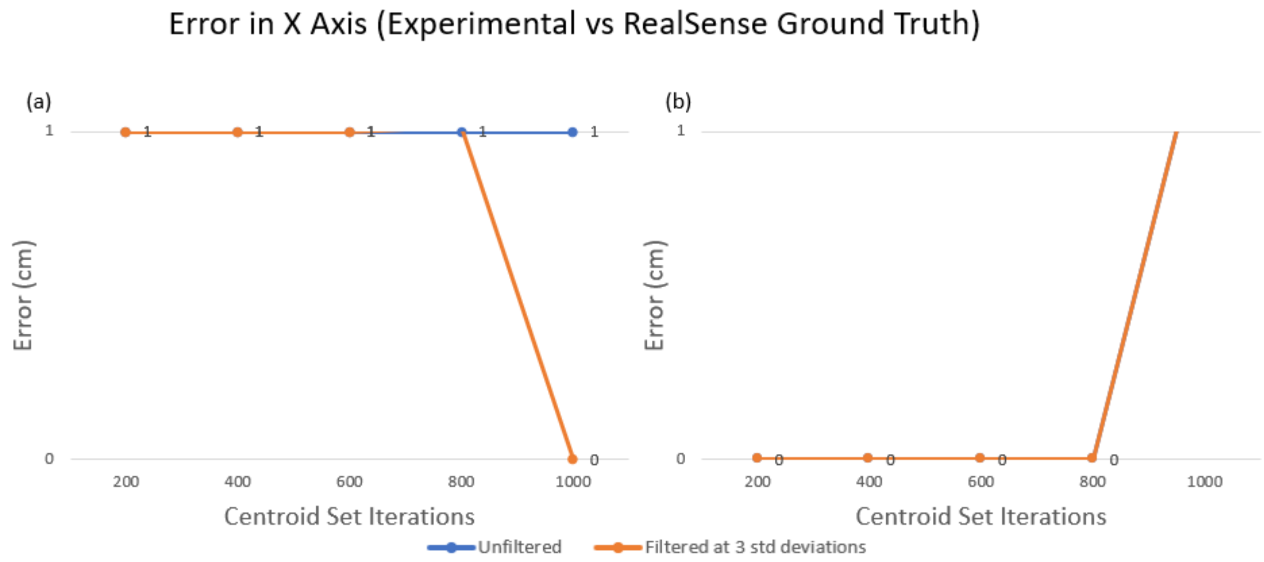

Figure 4.13. A graphical representation of the error in the *x* axis from the unfiltered data vs the filtered data at 3 standard deviations at 4m where (a) Sample size = 200 (b) Sample size = 1000

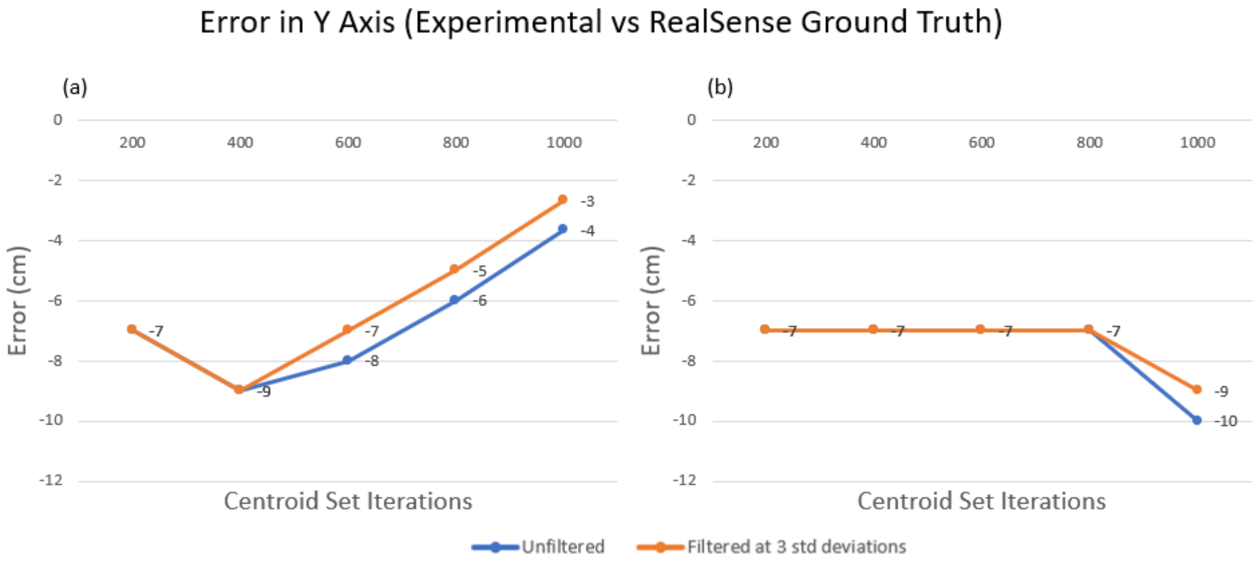

Figure 4.14. A graphical representation of the error in the *y* axis from the unfiltered data vs the filtered data at 3 standard deviations at 4m where (a) Sample size = 200 (b) Sample size = 1000

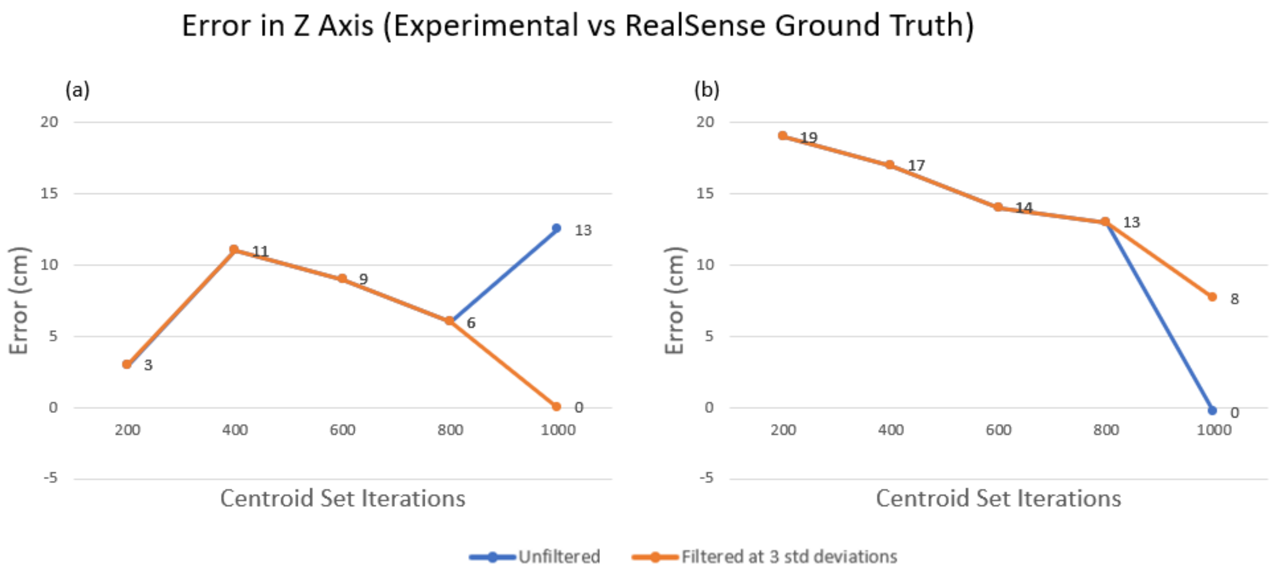

Figure 4.15. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 4m where (a) Sample size = 200 (b) Sample size = 1000

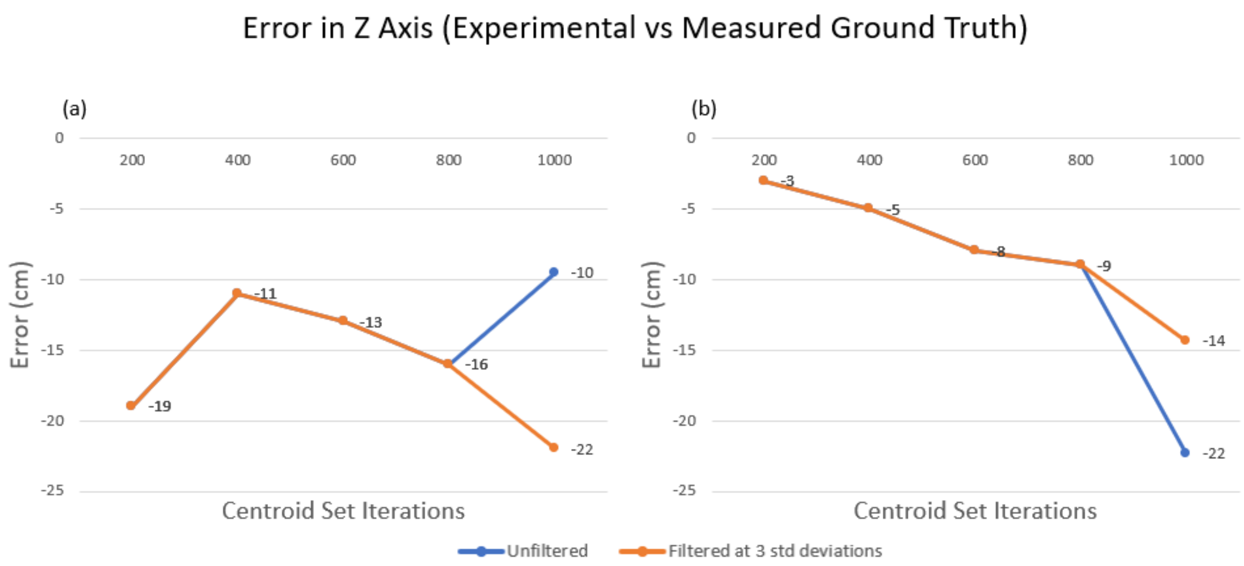

Figure 4.16. A graphical representation of the error in the *z* axis from the unfiltered data vs the filtered data at 3 standard deviations at 4m where (a) Sample size = 200 (b) Sample size = 1000

Unlike the results seen from 1m to 3m, both sample sets produced large errors. The largest errors were produced in the *z* axis in **Figure 4.16** of -22cm at both sample set sizes. In the sample set of 200, the filtering technique managed to reduce the error by as much as 13cm (depicted in **Figure 4.15**). However, in comparison to the filtering results produced from 1m to 3m, the filtering did not reduce as many large errors~~–~~, perhaps because of the high levels of noise at this distance.

4.1.5. Accuracy of *x*, *y* and *z* axes over all distances

Figure 4.17 below depicts the accuracy of the *x* axis over all the distances for both the unfiltered data and the filtered data.

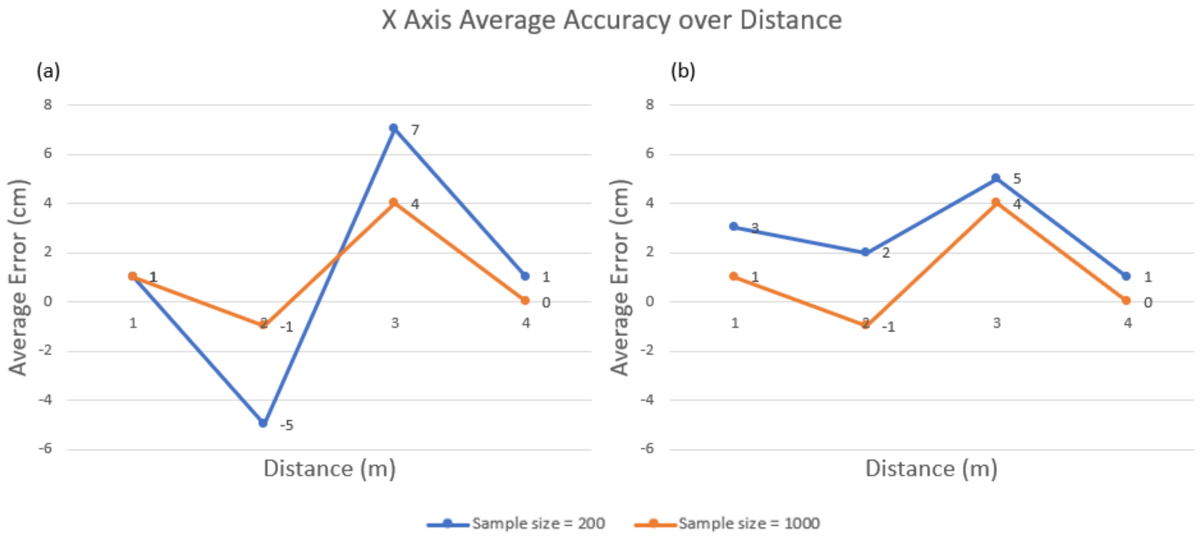

Figure 4.17. A graphical representation of the average error in the *x* axis at sample size 200 and 1000 over distance (a) unfiltered data (b the filtered data

From **Figure 4.17**, the largest average errors from both the unfiltered and filtered data came from the smaller sample size of 200. In **Figure 4.17** (a), 2m and 3m resulted in the worst errors, whereas in **Figure 4.17** (b), 3m showed the worst average errors. Comparatively, the larger sample size produced more accurate results over all the distances. For all the distances, 3m resulted in the largest average error. The filtering worked well in reducing the average error at shorter distances when the unfiltered error was larger (2m and 3m). Overall, the general offset in the error for the *x* axis seems to be to the right of the actual *x* centroid.

Figure 4.18 below depicts the accuracy of the *y* axis over all the distances for both the unfiltered data and the filtered data.

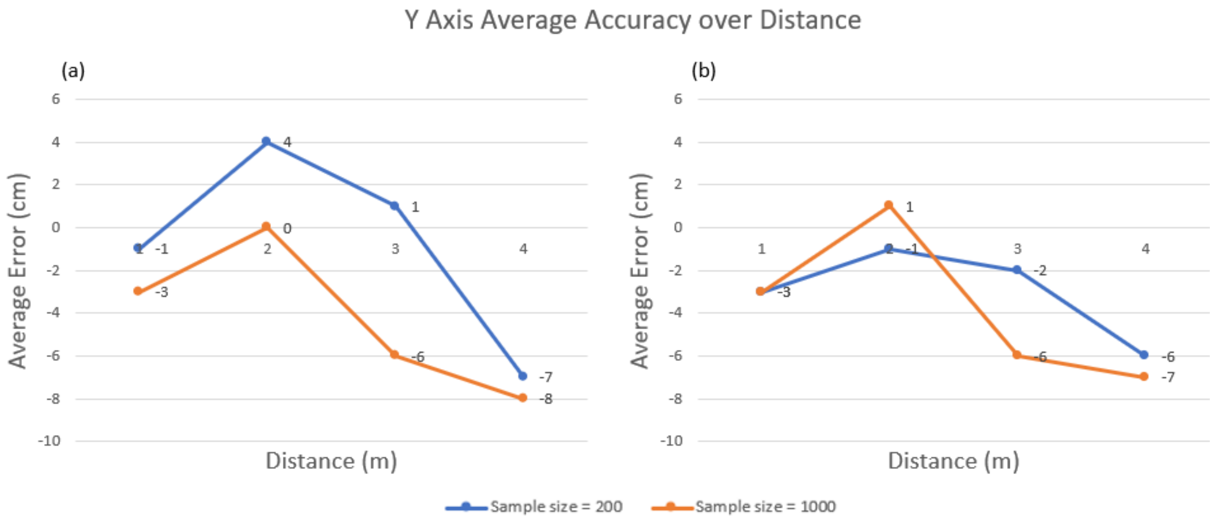

Figure 4.18. A graphical representation of the average error in the *y* axis at sample size 200 and 1000 over distance (a) unfiltered data (b the filtered data

From **Figure 4.18**, the largest average errors from both the unfiltered data (-8cm) and filtered data (-7cm) came from the larger sample size. In **Figure 4.18**, 3m and 4m resulted in the largest average errors. Comparatively, the smaller sample size produced more accurate results over all the distances, except for 2m in **Figure 4.18** (a). For all the distances, 4m resulted in the largest average error. The filtering worked well in reducing the average error at shorter distances (1m and 2m) when the error was larger and the general offset overall in the error for the experimental *y* values seem to be to lower than the actual *y* value (below the *y* centroid).

Figure 4.19 and **Figure 4.20** below depicts the accuracy of the *z* axis (compared with the RealSense Ground Truth and Measured Ground Truth respectively) over all the distances for both the unfiltered data and the filtered data.

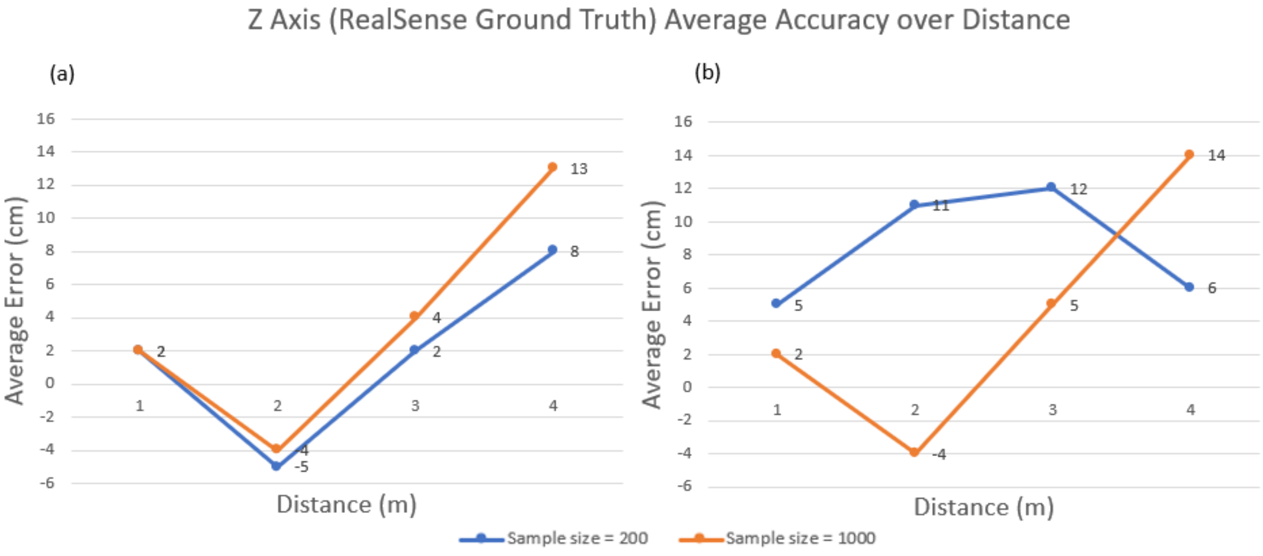

Figure 4.19. A graphical representation of the average error in the *z* axis at sample size 200 and 1000 over distance (a) unfiltered data (b the filtered data

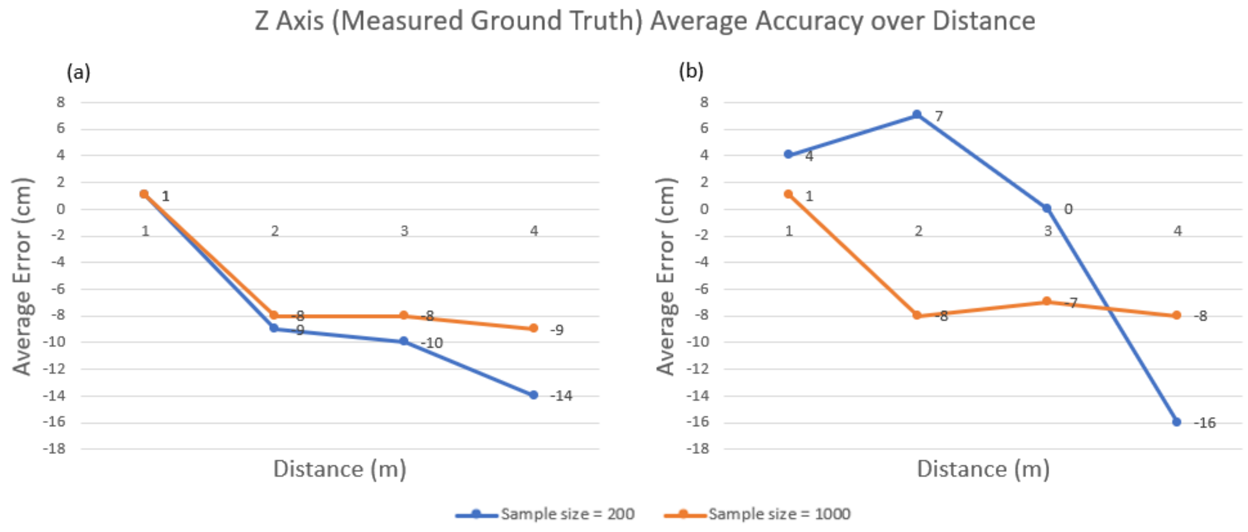

Figure 4.20. A graphical representation of the average error in the *z* axis at sample size 200 and 1000 over distance (a) unfiltered data (b the filtered data

From **Figure 4.19** and **Figure 4.20**, it is evident that the accuracy of the results of the *z* axis decreases as the ball moves further away from the D455. At 1m, the worst error in the *z* axis was 5cm, whereas from 2m onwards, this average error increased to -16cm at 4m away. The larger sample size produced more accurate results at the shorter distances (1m and 2m); however, from 3m onwards it had a negligible effect on the accuracy. Furthermore, the filter had a negligible effect on reducing the average error at further distances. The results from these two graphs showed significant differences in their depth value errors with distance. For the RealSense ground truth measurement, the average errors overestimated the *z* values (**Figure 4.19**), whereas the results from the tape measurement experiment underestimated the *z* values (**Figure 4.20**). For the tape measurement values, this could be due to human error, whereas a decrease in *z* accuracy is a known problem with triangulation-based systems like the RealSense, which have reduced depth accuracy with the square of the distance away (Grunnet-Jepsen & Tong, 2020**.** This depth noise can be due to multiple factors such as lighting, motion, or projector noise, which contribute to some depth noise in time. However, this noise can be improved by either running at lower frame rates and integrating the exposure longer, or by applying some temporal averaging to each pixel in the depth-map. The *z*-accuracy (up to 4 metres) of the D455 produces an accuracy of approximately 2% when measured against a flat surface that falls within 80% of its FoV (Intel, 2022). However, because the ball is not flat, this does create further inaccuracies for depth measurement, especially at further distances from the D455. Therefore, the reason for this decreasing accuracy in the *z* axis could be that as the ball moves further away from the camera, where the chance of collecting unwanted 3D coordinates will be greater as the camera will pick up coordinate values outside the area of the spherical ball. Subsequently, the large number of unwanted coordinates will produce a greater error in the centroid calculations. The Intel RealSense Viewer was used to obtain ground truth data for the *x* and *y* axes and the tape measure was used to get an estimate of the *z* distance from the camera. These tools provided a simple way to obtain data.

Between the two sampled sets, collecting more 3D data points (at sample set of 1000) reduced the number of large errors in the *x*, *y* and *z* axes at all the distances, especially distances further away from the camera (2m and 3m). Compared with sample size of 200, the sample size of 1000 proved to produce results with smaller errors, whereas the number of centroid set iterations had no effect on reducing the errors from either sample set. However, future work might look to explore this further by exploring how even larger sample sets might further improve accuracy.

It is evident from the analyses above that for distances beyond 2m the accuracy of the D455 for the *z* axis decreases significantly (**Figure 4.19** and **Figure 4.20**). Moreover, the most accurate results from all the axes were produced at 1m, from which the largest error produced was only 5cm in the *z* axis (**Figure 4.19** (b)). Therefore, conducting dynamic experiments beyond 2m would result in inaccurate results and lack of valid data from which to draw accurate conclusions, and as a result, the best region to obtain the most accurate results would be between 1m and 2m. Hence, this range was chosen to conduct the dynamic experiments on the ball, the results of which are detailed in section 4.2.

4.2. Dynamic Analysis

As mentioned in 3.2, the setup was designed to collect data from the ball in the *x*, *y* and *z* axes as it moved away from the D455 from which the results in the depth (*z* axis) and the XY plane were analysed.

The *z* axis experiments were conducted to test the ability of the camera to capture accurate centroid coordinates from an object moving away from it. This experiment used two distinct regions of *z* values (within the range of 20cm) to ensure that enough depth values were captured while the ball was moving away from the camera and to calculate the change in the ball's velocity at P_1. The two separate regions were chosen at distances within 2m from the camera as this region was the ideal region to capture the most accurate results. However, for the XY plane experiment, only 1m from the camera was the distance analysed as the vertical and horizontal displacements (XY plane) for the ball in the final experiment will be significantly less than the movement of the ball in the *z* axis. For both experiments, the distance between these two regions was chosen to eliminate the possibility of overlapping depth values from one region into another.

For the theoretical calculations, the equations of a projectile under gravity under the assumption of negligible drag were used; these assumptions are justified in Chapter 2.4. The equations to calculate the theoretical and experimental velocity at P_1 below were presented in Chapter 3.2 , but are repeated here for clarity.

<u>Theoretical equations:</u>

$$t_1 = \sqrt{\frac{2 \times P_1}{g}} \tag{7}$$

From this, the velocity at P_1 was calculated using equation **(7)**.

$$V_1 = gt_1 \tag{9}$$

<u>Experimental equations</u>:

$$(P_1 - P_0) = V_1 \Delta t \tag{27}$$

Manipulating **(5)** to calculate the velocity at P_1,

$$V_1 = \frac{(P_1 - P_0)}{\Delta t} \tag{28}$$

From which the error between the experimental and theoretical velocity was calculated using **(10)**.

$$V_{error} = V_{calculated} - V_{experimental} \tag{29}$$

Where:

g = 9.81 m/s^2
Δt = elapsed time in seconds for the ball to reach P_1 from P_0

Standard deviation was used as a filtering technique and compared with non-filtering results to analyse its effectiveness on reducing the error in the velocity for indoor and outdoor experiments. For both experiments, a sample set of 100 3D coordinates were collected at each region from which the average centroid was calculated from 200 centroid iterations (where 1 centroid iteration collected 1 set of 4 random 3D coordinates from the sample set). Using the first 100 3D coordinates allowed the most accurate 3D coordinates to be captured within each region in a certain timeframe. This method was then used to filter the values using standard deviation.

4.2.1. Indoor and Outdoor *Z* Axis Tests

For the *z* axis, the error in velocity between the theoretical calculations and the experimental results for the indoor (**Error! Reference source not found.**(a)) and outdoor tests (**Error! Reference source not found.**(b)). **Table 4.1** is used to compare the actual results from each test with their average, whereas **Figure 4.21** illustrates the trend in the error velocity from each test. For each test, the unfiltered results are compared with the filtered results at 3 standard deviations.

Table 4.1. Error in the velocity from the unfiltered and filtered data between 2 regions - Region 1: 0.6m – 0.8m, Region 2: 1.6m-1.8m for (a) Indoors, and (b) Outdoors

(a)

Test number	Error in Velocity (m/s)	
	Unfiltered	Filtered
1	1.43	1.95
2	-1.81	0.79
3	4.30	0.95
4	1.11	1.09
5	3.32	3.34
6	0.85	0.88
7	2.25	2.59
8	0.91	1.18
9	1.23	0.74
10	-1.59	-1.49
Average	1.20	1.20

(b)

Test number	Error in Velocity (m/s)	
	Unfiltered	Filtered
1	163064636.21	0.90
2	-1.93	0.60
3	3.29	4.39
4	2.72	3.99
5	102787504.49	-1.74
6	4.74	4.87
7	-10.30	-19.39
8	1.57	2.41
9	5.38	5.38
10	0.55	1.63
Average	26585214.67	0.30

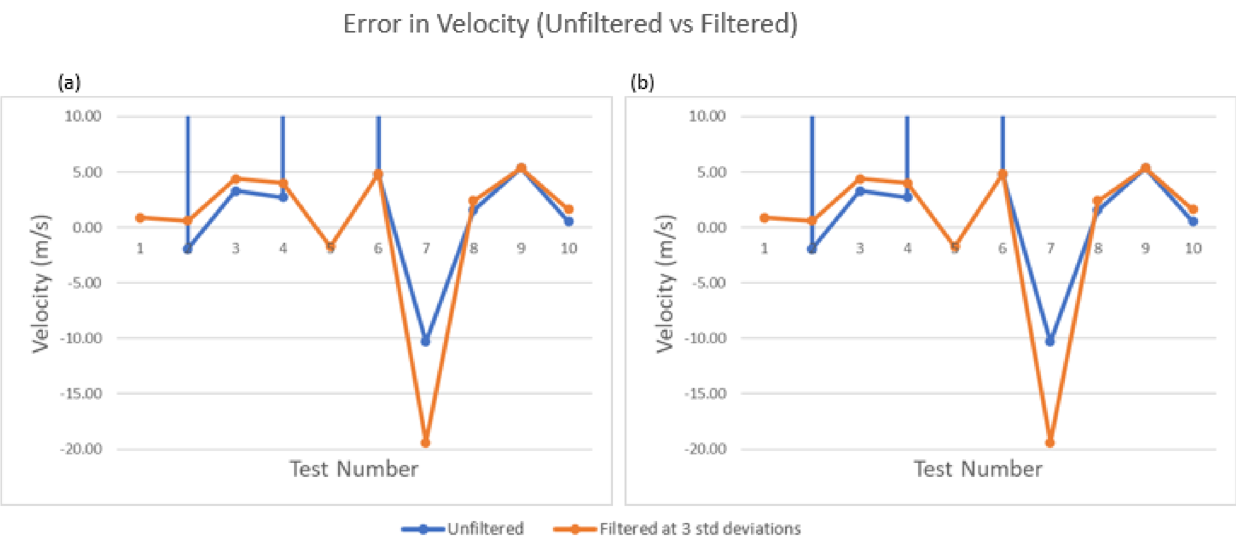

Figure 4.21. Error in the velocity from the unfiltered and filtered data between 2 regions - Region 1: 0.6m - 0.8m, Region 2: 1.6m - 1.8m for (a) Indoors, and (b) Outdoors

As is evident from **Table 4.1**, the filtered results (using 3 standard deviations) decreased the average error in velocity for the indoor and outdoor results. The filter especially helped with reducing the large error values in the velocities of the unfiltered data as is evident from test 1 and test 5 in **Error! Reference source not found.**(b). However, in some cases the filter did not reduce the error in velocity, which could be because the filter is more effective in filtering larger errors and therefore, when the unfiltered errors are smaller the filtered error values roughly output similar values that are either slightly smaller or slightly larger in comparison. In **Figure**

4.21 (b), the results of test 1 and test 5 are hidden as they would create difficulty in observing the smaller values in the graph – the reason thereof is discussed after Chapter 4.2.2 below.

4.2.2. Indoor XY Plane Test

The XY plane experiments were only conducted indoors and analysed by dividing the experiment into two. **Table 4.2** (a) outputs the error in the velocity between the two *y* regions (experiment depicted in **Figure 3.12**), which was analysed and calculated in the same way as the *z* axis experiment (depicted in **Figure 3.10**) (by dropping the ball vertically). Whereas **Table 4.2**(b) outputs the experimental change in displacement between these two regions compared with the theoretical change in displacement between these two regions.

Table 4.2. Error between the unfiltered and filtered data at 3 standard deviations at a 1m distance from the camera (a) in the *y* velocity (b) in the *x* displacement

(a)

Test number	Error in Velocity (m/s)	
	Unfiltered	Filtered
1	0.01	0.56
2	2.07	2.16
3	1.06	0.88
4	-0.04	0.31
5	1.69	1.94
6	5.46	2.93
7	2.91	1.66
8	3.00	2.16
9	0.95	0.91
10	1.85	1.57
Average	1.89	1.51

(b)

Test number	Error in Displacement (m)	
	Unfiltered	Filtered
1	-0.19	-0.06
2	0.53	0.20
3	0.36	0.00
4	-0.02	0.02
5	-1.19	0.15
6	-0.18	-0.09
7	-0.03	0.04
8	0.13	-0.04
9	0.16	0.17
10	-0.05	-0.02
Average	-0.05	0.04

As is evident from **Table 4.2**(a), the filter helped in reducing the average velocity error for both the filtered data. It also helped reduce large individual errors in the displacement as is evident from test number 3 and 5 in **Table 4.2** (b). However, it made a negligible impact in reducing the average error in the *x* displacement.

As the distance between the camera and the object increased, the accuracy of the depth information, including the *x* and *y* coordinates, decreased. According to (Brahmanage, 2019), the D435 RealSense depth camera works well for both indoor and outdoor environments. However, the short range and missing depth data pose challenges for the RealSense in outdoor environments. This is often due to a lack of associated depth data for matched RGB features which results in a low number of valid RGB feature matches. Intense lighting can result in gaps on the depth image due to low contrast in the infrared image. Furthermore, the problem of outdoor use is amplified by partial or no depth information from the camera. According to Condotta et al. (2020), the errors in the distance data originated from the common sources including, calibration errors, improper lighting, or image geometry, and smooth or bright surfaces. A further problem in capturing data in motion is motion blur. According to Marin et al. (2019), some of the flaws of an active stereo system in capturing depth data include elements with low reflectivity and external illumination. These factors could also be an explanation for the inaccuracy of the results taken outside. Richardson extrapolation estimates solution error using a series of simulations with systematically finer meshes and can be used to improve upon the launch velocity error estimation by obtaining an improved estimation of the uncertainty at the expense of needing more simulations. The values of lift and drag coefficients are evaluated to examine the effect of Reynolds number and of each time step and so yield a solution with higher accuracy from each time step pair. The two main aerodynamic parameters used in trajectory analysis are the Reynolds number (R*e*) and the spin parameter and is used to perform trajectory analysis in sports like tennis, baseball and soccer(J.E. Goff & Carre, 2010). It can also be a necessary consideration in this case where the ball is launched in the air with a certain trajectory and is discussed in more detail in Chapter 5.

4.3. Communication Analysis

Based on the communication method outlined in 3.3, the UDP protocol was the chosen communication protocol between the D455 and the HoloLens. Within this experiment, the HoloLens worked as the client that received 3D coordinate positions from the server running C++ code. A positive server-client communication was established through the ending and receiving of IP addresses of both the client and the server connection status. The sent 3D position data between the HoloLens and the server was outputted on the console for reference (**Figure 4.22**).

```
Initialising Winsock...Initialised.
Socket created.
Bind done
Waiting for data...
This message was received from 192.168.0.20 : Sent from HoloLens!
This message was sent to 192.168.0.20 : 0.000000,1.010000,2.020000,3.030000,4.040000
```

Figure 4.22. Console output of four random 3D coordinates received by the HoloLens from the server

Socket created established an open port between the server and the client and "bind done" indicated a successful connection to that port. The "Waiting for data" message served as a notification that the server is waiting to receive client data. The "Sent from HoloLens!" message was a message from the client through the IPv4 address indicated by 192.168.0.20 and the next line illustrated the string values sent to the HoloLens.

To ensure communication was working as expected, two tests were carried out. The first test analysed how the strings of values were received on the HoloLens and the second test sent a float value then delayed for six seconds before sending the second float value and was repeated for ten different values. The results showed that the 3D position strings were received correctly and in the correct order in all cases from both tests.

4.4. Calibration Analysis

Before this experiment was conducted, a communication was established between the D455 and the HoloLens using a UDP communication which was outlined in section 3.3. The main idea of the calibration procedure was to get the 3D point coordinates of the same points in different coordinate systems (HoloLens and D455), and then use these corresponding coordinates to calculate the transformation from one coordinate system to the other (D455 to HoloLens in this work). Once the calibration was complete, any 3D points collected by the D455 were transformed into the coordinate system of the HoloLens. Hence, accurate calibration information allows reprojection of data to a common reference system.

The results below use the transformation matrix determined from the calibration to transform known locations of each of the 3D centroid coordinates collected from the four calibration balls. This was completed to gain information about the error including the accuracy and precision, in the calibration. The known coordinate values (in m) for each ball are outlined below:

- Ball 1: (0,0,0.75)
- Ball 2: (0.05,0.2,1)
- Ball 3: (0.1,0.4,1.25)
- Ball 4: (0.15,0.6,1.5)

4.4.1. Mean Error Comparison Test

As a result, the position of each calibration ball (from the perspective of the HoloLens) is calculated ten times. Thereafter, the difference between the calculated values and the known value (fixed coordinates) for each ball were calculated from which the mean and the standard deviations were determined. The mean error between the transformed axes coordinates and the known axes values are presented in **Figure 4.23.** below.

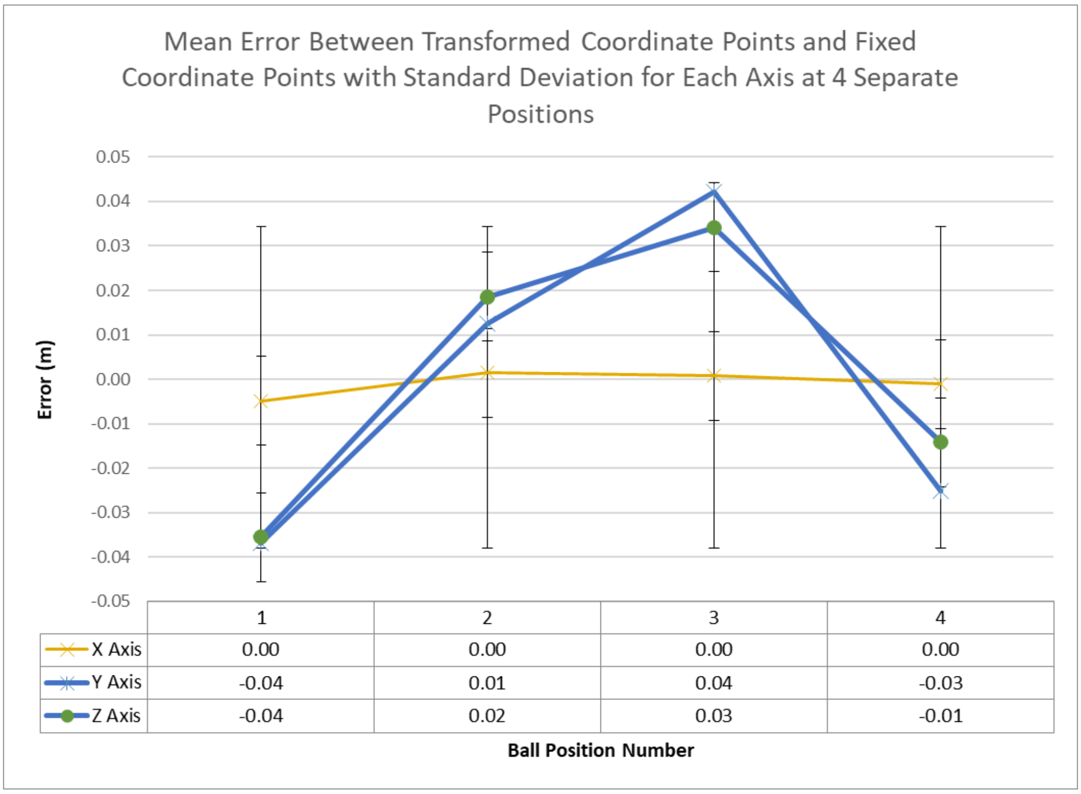

Figure 4.23. The mean error between the calculated axes values and the known axes values at the 4 calibration locations

As is evident from **Figure 4.23.**, the axes with the largest mean error from the four positions were the y and z axes. From these axes, the z axis and y axis produced the worst mean error of -4cm at position 1 (0.75m from the D455) respectively, whereas at position 3 (1.25m from the D455). the z axis produced the worst error of 4cm. Conversely, the x axis produced the most accurate results over all four positions with no error from the mean. Overall position 2 outputted more values closer to the mean for all the axes, while position 1 and 3 had the worst overall output errors. The error bars indicate one standard deviation at each position for each axis. As is evident from the graph, the calibrated *x* axis mean values gives the most accurate results as these values are close to zero. This could be because the ROI for the *x* axis was small, which resulted in the non-transformed values and the known values having small initial differences. In contrast, the *y* and *z* axes show the worst calibrated results further from the mean. The inaccuracy of these axes could be due to the initial offset of each axis, the inaccuracy of capturing the *y* and *z* coordinates as the ball moves further away and because of large differences between their initial, non-transformed values and their known values. Each of the mean error values fall within one standard deviation of the mean, which indicates an overall accurate calibration system. In an ideal situation, the mean and standard deviation would be zero, however this was not the case for these results.

4.5. 2D Prediction Trajectory Analysis

The purpose of this experiment was to design a system that could hit a desired target area at various distances between 1m and 2m. Before the 3D position data was sent to the HoloLens, a UDP connection was established between the server and client (discussed in Chapter 3.3). and the two coordinate axes (D455 and HoloLens) were calibrated using the ICP method (discussed in Chapter 3.4.1). Thereafter, the results of the accuracy of the trajectory simulation were collected at three separate distances:

- Box 1: 0.7m-1.2m
- Box 2: 1.2m-1.7m
- Box 3: 1.7m-2.2m

The same method of capturing 2 separate coordinates in two separate regions (mentioned in Chapter 3.2) was used for this experiment. For each box, the capturing regions were adjusted to suit the distances at which each box was placed. The Unity script ran alongside the C++ server code to enable 3D position data to be sent to the HoloLens. For launching the ball, the system was first calibrated from which the ball's trajectory was simulated using Unity's in-built physics engine.

4.5.1. Box Hits Versus Shot Distance

The results from the proportion of box hits vs the distance of the shot (with a line for each of the) are plotted in **Figure 4.24** below. All distances showed a statistically significant difference between the real and virtual worlds which increased as the distance became larger (p values of 0.009, 1.31×10^{-4} 1.59×10^{-5} in order of increasing distance to target). Thirty shots are taken at each distance and the percentage of real-world number of box hits versus virtual world box hits are recorded.

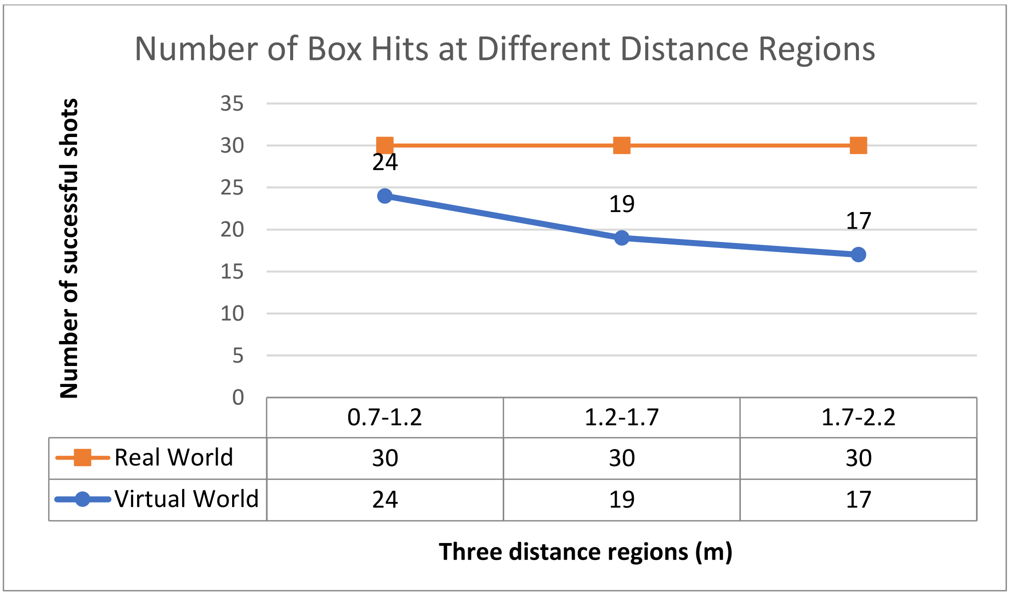

Figure 4.24. Number of successful box hits over a sample size of 30 shots performed at each distance range

From **Figure 4.24**, the number of successful virtual box hits decreased as the distance of the box moved further away from the D455. In the real and virtual world, the boxes were centred around $y = 0$ and therefore had 0.125m on either side of the y axis for potential horizontal displacement of the ball. This large area was also chosen to compensate for any error in the z axis. Some of the x coordinates at P_1 were outside the target area, which resulted in the ball missing the target areas.

Noisy data was picked up during the experiment from collecting excess position data from the ROI, which also affected the capturing of accurate coordinate values. Hence, a 6 second delay timer was used between collecting two position coordinates between each throw as this cleared the excess position data from the ROI prior to each launch experiment. Inaccurate calibration would also affect the accuracy of P_0 and P_1, which then adds to the inaccuracies of the z values that occur at distances further away from the camera and confirms the low accuracy of balls hitting the target in this region. As the ROIs created for each distance progressively becomes smaller at further distances from the camera, the accuracy of throwing the ball through those regions decreases, which then increases the likelihood of not collecting 3D data within those regions. As a result, tests must be rerun more frequently. From the dynamic experiments conducted in section 3.2, the regions with the least error in velocities proved to be between 0.7m and 1m and 1.3m-1.6m. Therefore, it makes sense that these 2 regions had the most successful results.

4.5.2. Ball Trajectory Paths

Figure 4.25 below illustrates a mathematical prediction of a typical shot trajectory of a free throw showing the impact of the launch error on the prediction of the ball path. The chosen examples of an ideal free throw launch velocity, launch height and launch angle were 7.47m/s, 2.1m and 53 degrees respectively taken from the free throw line 4.57m away from the release of the ball (Okubo & Hubbard, 2006). The average launch error in the z axis was 1.2m/s, which was taken from the average filtered error from the indoor experiment. And the average launch error velocity in the y axis was 1.51m/s taken from the average filtered error. The equations used to calculate the undershoot and overshoot trajectory paths in Chapter 3.5.1 are re-presented below for clarity.

For the ideal trajectory:

- z axis

$$z = z_0 + V_z t \tag{23}$$

- y axis

$$y = y_0 + V_y t - 0.5gt^2 \tag{24}$$

For the undershoot and the undershoot trajectories:

- z axis

$$z = z_0 + (V_z \pm e_z)t \tag{25}$$

- y axis

$$y = y_0 + (V_y \pm e_y)t - 0.5gt^2 \tag{26}$$

Where:

- z_0 - Initial horizontal position
- y_0 - Initial vertical position
- V_z – Ideal velocity in the z axis
- V_y – Ideal velocity in the y axis
- e_z – Error velocity in the z axis
- e_y – Error velocity in the y axis
- g = 9.81m/s^2

The diameter of a regular-sized basketball hoop is 46cm and its height is 3.05m above the ground, which is depicted below as a horizontal yellow line. Three standard deviations was used when calculating the launch velocity errors.

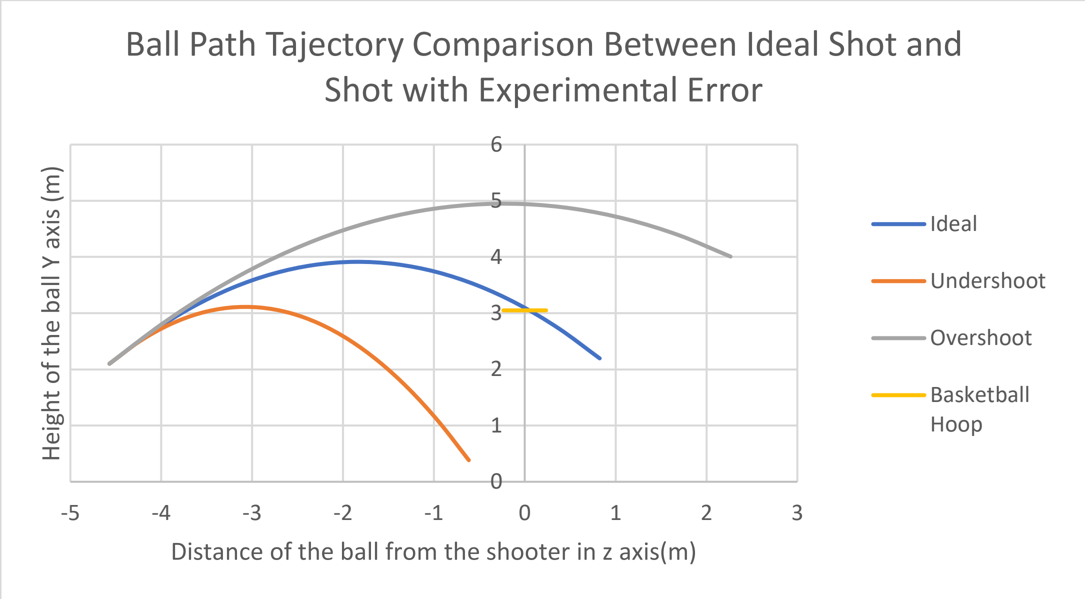

Figure 4.25. A comparison between the ideal ball bath trajectory and the experimental ball path trajectory that uses the average launch velocity errors (using 3 standard deviations)

From **Figure 4.25**, the error velocity in each axis was used to determine the undershoot and overshoot trajectories. Only the ideal path produced a trajectory path that resulted in the ball going through the hoop, whereas both the undershoot and overshoot trajectories resulted in the ball missing the hoop by metres. Based on the inaccuracies of the D455 beyond the ideal range of 2m and the challenge with capturing accurate centroid coordinates of a ball in motion, it is evident that overall, the method would perform poorly for the free throw. The goal of the future work is to attempt to tighten these bands as much as possible. Additionally, this result would only worsen if the calibration errors were also considered. However, these errors were ignored as they would have a comparatively small effect in comparison to the launch velocity errors.

For reference, the *z* and *y* error velocities were adjusted manually to see what the maximum allowed *z* and *y* launch errors were that resulted in the ball passing through the hoop. These values were 0.07m/s for both the *z* and *y* axes, which are illustrated in **Figure 4.26**.

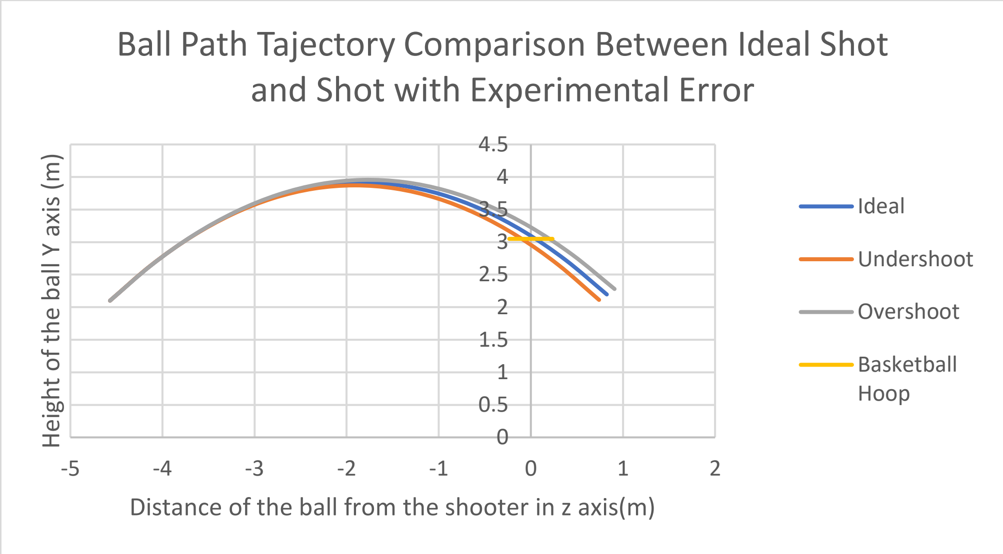

Figure 4.26. A comparison between the ideal ball bath trajectory and the experimental ball path trajectory that uses the average launch velocity errors

As is evident from **Figure 4.26**, the maximum allowed launch velocity error for both axes that would result in the ball passing through the hoop was 0.07 m/s. This would mean reducing the launch velocity errors in the *z* axis by 1.13m/s and the *y* axis by 1.44 m/s to fall within the maximum allowed error range. However, this was not achievable with this current experimental setup and required significant improvements in collecting coordinates of a ball moving away from the D455. This could be done using better camera technology or by placing multiple cameras in the scene that could capture the different views (front on and side on). These potential improvements are discussed in more detail in Chapter 5.

Chapter 5 Conclusion and Recommendations

The main objective at the start was to design an AR basketball training aid that would accurately track the location of the ball at a point in real space from which a ball would spawn and its ball path trajectory simulated. To achieve this objective, it was decided to combine the object tracking capabilities from the D455 and the AR functionalities of the HoloLens. One of the challenges in this work was ensuring that accurate 3D position data could be obtained from an object moving away from the D455. To achieve this, two sample sets of 3D points from two distinct regions were collected from which an algorithm was used to calculate the centroid from each region along with their timestamp. However, it was found that distances beyond 2m from the camera resulted in inaccurate depth values and as a result, tracking the ball was restricted to within the ideal 2m range. Furthermore, standard deviation was used to filter out any additional noisy values that occurred within these regions. This filtering technique had mixed success in reducing the amount of noisy 3D data. In some cases where there was a large deviation from the mean, the filter was successful in reducing noisy data; however, as the distance from the camera increased, the filter became less effective in this regard. This was mainly seen in the static tests at distances beyond 2m and mainly affected the values in the z axis.

The ICP algorithm was used to transform the 3D coordinates taken from the D455 axis and align it with the HoloLens axis. These calibration experiments validated this system; however, there were some inaccuracies in the transformed axis that arose due to using a limited amount of calibration points. Hence, not utilising the full effect of the algorithms intended use (for large data sets). On average, the worst calibration results for all the axes were at position 1 (0.75m) and position 4 (1.5m); whereas the most accurate results were at position 2 (1m) from the D455. This could be due to the inaccuracy of the camera in picking up coordinate data at distances too close and too far away.

For this system to work effectively UDP communication was chosen to send 3D position data to the HoloLens, which worked effectively as a simple way to send the desired 3D coordinates quickly. Unity was the chosen game engine through which the HoloLens received the required 3D position data and worked successfully in achieving this objective.

As is evident from the final 2D trajectory experiment, the z and y launch velocity errors at 4.57m away resulted in the ball missing the hoop by a few metres. This result was to be expected as the dynamic experimental errors within the ideal 2m distance resulted in z and y launch errors of 1.2m/s and 1.51 m/s respectively. Furthermore, the error in t1 is not considered which could also account for the inaccuracies in the velocity calculation. This could be improved by further analysing and refining how the time is captured at each position. For the ball to pass through the hoop at 4.57m away, the maximum allowed error in velocity would need to be reduced significantly. Therefore, the initial objective of achieving object tracking using a ball simulation at this distance was unsuccessful. However, this experiment was adjusted to hit targets of a certain area at three different distances and in this respect, the desired objective of simulating a ball trajectory and hitting a target, was achieved.

Although this simplified system partially achieved its desired outcome, there were a few areas which could be improved upon to increase the accuracy of this system. Because this system was designed to get the 3D position of the object in front of it, this created challenges in ensuring that the correct position data was being sent through. One option for this would be to utilise better 3D tracking algorithms or filtering techniques such as template matching, Kalman filter, and particle filter, tracking and 3D reconstruction from different views or use physics-based algorithms. These could give access to additional dynamic parameters that would assist in ball tracking. In addition to better filtering techniques, larger 3D data sets could be collected at each region when conducting dynamic experiments to increase the probability of collecting valid data within each region. Another statistical approach that could improve the accuracy of the system, is

Richardson’s extrapolation (mentioned in section 4.2). In this case, Richardson extrapolation can be used to better predict the ball’s trajectory by estimating the aerodynamic effects of the ball (such as lift and drag) at refined increments along its path. This can be done by minimising the least-squares deviation between the data (at refined increments) and the computational solution of an equation (J.E. Goff & Carre, 2010). One could also use multiple tracking cameras positioned at different locations to enable the 3D data from the ball to be more accurately mapped. Having one camera tracking the ball from the side and another tracking the ball moving forward would enable the system to assess all the angles and trajectories that the ball could take in a basketball shot. These cameras could then be used in combination to feedback accurate 3D position results to the HoloLens. Using better camera technology that can deliver images at better resolution and speed (frame rate) over long distances and accommodate varying illumination conditions, could have further improved the tracking ability of this system. These systems may be relatively cumbersome to set up and calibrate and carry additional hardware expense (hence the original intent of this thesis to explore the potential of simpler systems).

In an ideal world, it would have been best to use the HoloLens as a solo tracking device, which would limit the overhead of using external tracking devices to send through 3D position data. In hindsight, maybe the close-range mode would have been suitable, given what was ultimately found with the RealSense. However, this tracking technology in the HoloLens is still novel, so the use of accurate external tracking devices, such as the Azure Kinect, may be a future avenue of research to achieve more accurate tracking data, which could in turn could be used to achieve more accurate tracking on the HoloLens. Like the RealSense D455 device, it has object tracking capabilities with the difference being that it uses a ToF depth sensor which is said to improve tracking objects in dim light conditions as well as the processing power to support longer distance ranges (Lakshan, 2020). An alternative approach to using a camera for object tracking, could be to use smart balls with IMU sensors in them. This would be able to give accurate sensor data of the ball in mid-air, which could then be streamed to the HoloLens to track real-time data such as optimal shots and shot count (Chua, 2018). However, this could damage the sensor over time and may not be the most sustainable approach to take. This thesis has demonstrated that the proposed approach is at least feasible and will likely become more viable by using improved tracking technologies such as those suggested here, those developed in future research, or even the onboard tracking capabilities of future iterations of AR devices like the HoloLens which are likely to continue to improve.

Chapter 6 References

Andrade, E., Woods, J., & Ghanbari, M. (2005). Region-based analysis and retrieval for tracking of semantic objects and provision of augmented information in interactive sport scenes. *IEEE Transactions on Multimedia, 7*(6), 1084-1096. doi:10.1109/TMM.2005.854417

Asvadi, A., Mahdavinataj, H., Karami, M., & Baleghi, Y. (2014). *Incremental Discriminative Color Object Tracking*. Paper presented at the International Symposium on Artificial Intelligence and Signal Processing (AISP).

Bahri, H., Krčmařík, D., & Kočí, J. (2019). *Accurate Object Detection System on HoloLens Using YOLO Algorithm*. Paper presented at the 2019 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO).

Barla, N. (2022). The Complete Guide to Object Tracking [+V7 Tutorial]. Retrieved from https://www.v7labs.com/blog/object-tracking-guide

Bourke, P. (2002). Equation of a Sphere from 4 Points on the Surface. Retrieved from http://paulbourke.net/geometry/circlesphere/

Brahmanage, G. L., H. (2019). *Outdoor RGB-D Mapping Using Intel-RealSense*. Paper presented at the 18th IEEE Sensors Conference.

Bryght. (2022). Hololens-2. In.

Chua, J. (2018). Developing IMU Sensors For Capturing Motion In Sports. Retrieved from https://sportstechnologyblog.com/2018/08/04/developing-imu-sensors-for-sports/

Condotta, I. C. F. S., Brown-Brandl, T. M., Pitla, S. K., Stinn, J. P., & Silva-Miranda, K. O. (2020). Evaluation of low-cost depth cameras for agricultural applications. *Computers and Electronics in Agriculture*. doi:10.1016/j.compag.2020.105394

da Silva, A. M., Albuquerque, G. S. G., & Alencar de Medeiros, F. P. (2021, JUN 23-26, 2021). *A Review on Augmented Reality applied to Sports*. Paper presented at the 16th Iberian Conference on Information Systems and Technologies (CISTI).

Ding, P., & Song, Y. (2015). *Robust Object Tracking using Color and Depth Images with a Depth based Occlusion Handling and Recovery*. Paper presented at the 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD).

Evans, G., Miller, J., Pena, M. I., MacAllister, A., & Winer, E. (2017). *Evaluating the Microsoft HoloLens through an augmented reality assembly application*. Paper presented at the Mechanical Engineering Conference Presentations, Papers, and Proceedings.

Farasin, A., Peciarolo, F., Grangetto, M., Gianaria, E., & Garza, P. (2020). *Real-time Object Detection and Tracking in Mixed Reality using Microsoft HoloLens*. Paper presented at the 15th International Conference on Computer Vision Theory and Applications.

Fukuda, T., Zhu, Y., & Yabuki, N. (2018). *Point Cloud Stream on Spatial Mixed Reality Toward Telepresence in Architectural Field*. Paper presented at the 36th International Conference on Education and Research in Computer Aided Architectural Design in Europe (ECAADE).

Garon, M., Boulet, P.-O., Doiron, J.-P., Beaulieu, L., & Lalonde, J.-F. (2016). *Real-Time High Resolution 3D Data on the HoloLens.* Paper presented at the IEEE International Symposium on Mixed and Augmented Reality (ISMAR-Adjunct).

Giancola, S., Sala, R., & Valenti, M. (2018). *A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy TechnologiesSpringerBriefs in Computer Science*, (1st ed., pp. 1 online resource (IX, 90 pages 101 illustrations, 188 illustrations in color). doi:10.1007/978-3-319-91761-0

Giancola, S., Zarzar, J., & Ghanem, B. (2019). *Leveraging Shape Completion for 3D Siamese Tracking*. Paper presented at the IEEE Conference on Computer Vision and Pattern Recognition.

Goff, J. E. (2013). A review of recent research into aerodynamics of sport projectiles. *Sports Engineering, 16*(3), 137-154. doi:10.1007/s12283-013-0117-z

Goff, J. E., & Carre, M. J. (2010). Soccer ball lift coefficients via trajectory analysis. *European Journal of Physics, 31*(4), 775-784. doi:10.1088/0143-0807/31/4/007

Grunnet-Jepsen, A., & Tong, D. (2020). Depth Post-Processing for Intel® RealSense™ Depth Camera D400 Series. Retrieved from https://dev.intelrealsense.com/docs/depth-post-processing

Gupta, A., Zafar, W., & Schneiders, D. (2018). *Spatial Reconstruction using Microsoft HoloLens*. Retrieved from

Hamilton, G., & Reinschmidt, C. (1997). Optimal trajectory for the basketball free throw. *Journal of Sports Science*, 491-504. doi:10.1080/026404197367137

Han, M., & Kim, I. (2013). *Hue Modeling for Object Tracking in Multiple Non-overlapping Cameras.* Paper presented at the 7th Multi-Disciplinary International Workshop on Artificial Intelligence (MIWAI).

Huang, Y. C., Liao, I.N. , Chen, C.H. , Ik, TIU & Peng, W.C. (2019). *TrackNet: A Deep Learning Network for Tracking High-speed and Tiny Objects in Sports Applications.* Paper presented at the 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS).

Hübner, P., Clintworth, K., Liu, Q., Weinmann, M., & Wursthorn, S. (2020). Evaluation of HoloLens tracking and depth sensing for indoor mapping applications. *Sensors, 20*(4), 1021.

Intel. Intel D455 Depth Camera, 0.4m Min Depth Distance, 20m Max Range. In: RS Components.

Intel. (2019). Projection in Intel RealSense SDK 2.0. Retrieved from https://dev.intelrealsense.com/docs/projection-in-intel-realsense-sdk-20

Intel. (2022). Intel® RealSense™ Product Family D400 Series. Retrieved from chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://www.intelrealsense.com/wp-content/uploads/2022/05/Intel-RealSense-D400-Series-Datasheet-April-2022.pdf

Itoh, Y., Orlosky, J., Kiyokawa, K., & Klinker, G. (2016, FEB 25-27, 2016). *Laplacian Vision: Augmenting Motion Prediction via Optical See-Through Head-Mounted Displays*. Paper presented at the 7th Augmented Human International Conference (AH).

K. Hung, G., & Macari Pallis, J. (2004). *In Biomedical Engineering Principles in Sports*: Springer, Boston, MA.

Kamel, H. (2021). Getting Started with the RealSense™ D455. Retrieved from https://nz.mouser.com/applications/getting-started-with-realsense-d455/

Ku, T., Bi, D., Zha, Y., Zhang, L., & Yang, Y. (2015). *Corlor Names based scale-adaptive object tracking*. Paper presented at the Chinese Automation Congress (CAC).

Lakshan. (2020). ToF (Time of Flight) vs Stereo Vision – 3D Imaging Technology Comparison. Retrieved from https://www.seeedstudio.com/blog/2020/07/03/3d-imaging-technology-comparison-tof-time-of-flight-vs-stereo-vision-m/

Liu, Y., Jing, X.-Y., Nie, J., Gao, H., Liu, J., & Jiang, G.-P. (2019). Context-Aware Three-Dimensional Mean-Shift With Occlusion Handling for Robust Object Tracking in RGB-D Videos. *IEEE, 21*.

Marin, G., Agresti, G., Minto, L., & Zanuttigh, P. (2019). A multi-camera dataset for depth estimation in an indoor scenario. *Data In Brief*. doi:10.1016/j.dib.2019.104619

Microsoft. (2022a). About HoloLens 2. Retrieved from https://docs.microsoft.com/en-us/hololens/hololens2-hardware

Microsoft. (2022b). All you need to know on Azure Kinect. In.

Microsoft. (2022c). Azure Kinect DK. Retrieved from https://azure.microsoft.com/es-es/services/kinect-dk/#features

Microsoft. (2022d). hololens-2-industrial-edition. In: Microsoft.

Microsoft. (2022e). HoloLens 2. Retrieved from https://www.microsoft.com/en-us/hololens/hardware

Microsoft. (2022f). HoloLens 2 Exploded View Diagram. In: Microsoft.

Microsoft. (2022g). HoloLens 2 Front View. In: Microsoft.

Microsoft. (2022h). HoloLens Research Mode. Retrieved from https://docs.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/research-mode

Microsoft. (2022i). Microsoft HoloLens Development. Retrieved from https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/XRDevelopment/AR/ARPlatforms/HoloLens/

Microsoft. (2022, January 1). What is Mixed Reality Toolkit 2? Retrieved from https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2022-05#documentation

Moon, S., Lee, J., Nam, D., Yoo, W., & Kim, W. (2018). *A Comparative Study on Preprocessing Methods for Object Tracking in Sports Events.* Paper presented at the 20th International Conference on Advanced Communication Technology (ICACT).

Okubo, H., & Hubbard, M. (2006). Dynamics of the basketball shot with application to the free throw. *Journal of Sports Science, 24*(12), 1303-1314. doi:10.1080/02640410500520401

Przednowek, K., Krzeszowski, T., Przednowek, K. H., & Lenik, P. (2018). A System for Analysing the Basketball Free Throw Trajectory Based on Particle Swarm Optimization. *Applied Sciences*. doi:10.3390/app8112090

Qi, H., Feng, C., Cao, Z., Zhao, F., & Xiao, Y. (2020). *P2B: Point-to-Box Network for 3D Object Tracking in Point Clouds.* Paper presented at the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR).

Rajkumar, A. (2021). Point Set Registration Methods — Discussion and Implementation. Retrieved from https://medium.com/swlh/point-set-registration-methods-discussion-and-implementation-58efa98d0b68

RealSense, I. (2021). Intel® RealSense™ Camera Depth Testing Methodolgy. Retrieved from https://dev.intelrealsense.com/docs/camera-depth-testing-methodology

Rivlin, E., Rudzsky, M., Goldenberg, R., Bogomolov, U., & Lepchev, S. (2002). *A real-time system for classification of moving objects.* Paper presented at the 16th International Conference on Pattern Recognition.

Runze, W., Yuxing, W., & Jianlin, Z. (2017). *Improved Object Tracking Algorithm Based on Tracking-Leaning-Detection Framework.* Paper presented at the International Conference on Electronics Information and Emergency Communication (ICEIEC).

Slavcheva, M., Kehl, W., Navab, N., & Ilic, S. (2017). SDF-2-SDF Registration for Real-Time 3D Reconstruction from RGB-D Data. *International Journal of Computer Vision (2018)*.

Soltani, P., & Morice, A. H. P. (2020). Augmented reality tools for sports education and training. *Computers and Education*, 2.

Spalding. (2022). U-Turn™ 54 Inch Acrylic In-Ground. In: Spalding.

Stark, E., Kucera, E., Haffner, O., Drahos, P., & Leskovsky, R. (2020). Using Augmented Reality and Internet of Things for Control and Monitoring of Mechatronic Devices. *Electronics, 9*(8). doi:10.3390/electronics9081272

Swalaganata, G., Muniri, & Affriyenni, Y. (2018). *Moving Object Tracking Using Hybrid Method*. Paper presented at the International Conference on Information and Communications Technology.

Tran, C. M., & Silverberg, L. M. (2008). Optimal release conditions for the free throw in men's basketball. *Journal of Sports Science, 26*(11), 1147-1155. doi:10.1080/02640410802004948

Unity. (2021). Unity. Retrieved from https://unity.com/products/unity-platform

Wu, Y., Lim, J., & Yang, M.-H. (2013). Online Object Tracking: A Benchmark. *Computer Vision Foundation*.

Wu, Z., Zhao, T., & Nguyen, C. (2020). *3D Reconstruction and Object Detection for HoloLens*. Paper presented at the 2020 Digital Image Computing: Techniques and Applications (DICTA).

Zhou, F., Duh, H. B.-L., & Billinghurst, M. (2008). *Trends in Augmented Reality Tracking, Interaction and Display: A Review of Ten Years of ISMAR.* Paper presented at the 7th IEEE International Symposium on Mixed and Augmented Reality.

APPENDIX A

Collect a sample set from a ROI

```
int main()
{
// Create a Pipeline - this serves as a top-level API for streaming and
processing frames
    rs2::pipeline p;

    // Configure and start the pipeline
    p.start();

    auto start = chrono::steady_clock::now();

    //keep listening for data
    while (1)
    {
        // Block program until frames arrive
        rs2::frameset frames = p.wait_for_frames();

        // Try to get a frame of a depth image
        rs2::depth_frame depth = frames.get_depth_frame();

        // Get intrinsic camera parameters
 auto intrins = rs2::video_stream_profile(depth.get_profile()).get_intrinsics();

        // Get the depth frame's dimensions
        auto width = depth.get_width();
        auto height = depth.get_height();

       // Create a Region of Interest for RealSense using a nested for loop
        for (int x = (width / 2) - 50; x <= (width / 2) + 50; x++)
        {
            for (int y = (height / 2) - 200; y <= (height / 2) + 150; y++)
            {
                const float pixel[] = { (float)x, (float)y };
                float distance = depth.get_distance(x, y);
                float planarPoint[3];

                // Looks in specified distance region
                if (0.7 < distance && distance < 0.8)
                {
                    // Converts pixel coordinates to 3d point coordinates
           rs2_deproject_pixel_to_point(planarPoint, &intrins, pixel, distance);

                    // Pushes new coordinates onto the end of g1 vector
       g1.push_back(make_tuple(planarPoint[0], planarPoint[1], planarPoint[2]));

    //g1 is sample set of 3D points
                    if (g1.size() == 1000)
                    {
                        //Calculate Centroid method
                        Centroid(g1);
                    }

                }
            }
        }
```

Figure A 1. Code that collects a sample set within a ROI

Centroid Calculation

```
// Centroid method gets the position of the centroid at 4 separate positions as
input for calibration
void Centroid(vector<tuple<float, float, float>>v)
{
    //count refers to the number of different 4 random coordinates chosen from   the sample set
    for (int j = 0; j < count1; j++)
    {
        // Erase any duplicates from the vector
        v.erase(unique(v.begin(), v.end()), v.end());

        // Takes in 4 random xyz coordinates
        for (int i = 0; i < 4; i++)
        {
            random_shuffle(v.begin(), v.end());

            //Iterates through the input vector
            for (auto& it : v)
            {
                p[i].x = get<0>(it);
                p[i].y = get<1>(it);
                p[i].z = get<2>(it);
            }
        }

        //Find determinant m11
        for (int i = 0; i < 4; i++)
        {
            a[i][0] = p[i].x;
            a[i][1] = p[i].y;
            a[i][2] = p[i].z;
            a[i][3] = 1;
        }
        m11 = Determinant(a, 4);
        sum1 += m11;

        //Find determinant m12
        for (int i = 0; i < 4; i++)
        {
            a[i][0] = p[i].x * p[i].x + p[i].y * p[i].y + p[i].z * p[i].z;
            a[i][1] = p[i].y;
            a[i][2] = p[i].z;
            a[i][3] = 1;
        }
        m12 = Determinant(a, 4);
        sum2 += m12;

        //Find determinant m13
        for (int i = 0; i < 4; i++)
        {
            a[i][0] = p[i].x * p[i].x + p[i].y * p[i].y + p[i].z * p[i].z;
            a[i][1] = p[i].x;
            a[i][2] = p[i].z;
            a[i][3] = 1;
        }
        m13 = Determinant(a, 4);
        sum3 += m13;

        //Find determinant m14
        for (int i = 0; i < 4; i++)
        {
            a[i][0] = p[i].x * p[i].x + p[i].y * p[i].y + p[i].z * p[i].z;
            a[i][1] = p[i].x;
            a[i][2] = p[i].y;
            a[i][3] = 1;
        }
        m14 = Determinant(a, 4);
        sum4 += m14;

        // Calculate the average of each determinant (m11,m12,m13,m14,m15)
        float ave1 = sum1 / count1;
        float ave2 = sum2 / count1;
        float ave3 = sum3 / count1;
        float ave4 = sum4 / count1;

        c[j].x = 0.5 * ave2 / ave1;
        c[j].y = (Yh - (-0.5 * ave3 / ave1));
        c[j].z = 0.5 * ave4 / ave1;

        sum_cx += c[j].x;
        sum_cy += c[j].y;
        sum_cz += c[j].z;
    }
}
```

Figure A 2. Code for calculating the centroid from a sample set

Covariance Matrix Calculation

```
/******************************************************************************
Function : CalcualateICPCovarianceMtx
Description: Calculate covariance between the two mass-points
Input1 : Size of the point cloud
Input2 : Data mass-point - perspective of the realsense
Input3 : Model mass-point - perspective of the hololens
Returns : Covariance Matrix
******************************************************************************/

Mat CalcualateICPCovarianceMtx(int length, vector<Point3d>& centerPCL,
vector<Point3d>& centerMCL) // PCL - Realsense, MCL - Hololense
{
    Mat covariance = Mat::zeros(3, 3, CV_64F);

    double sumXX = 0., sumXY = 0., sumXZ = 0.;
    double sumYX = 0., sumYY = 0., sumYZ = 0.;
    double sumZX = 0., sumZY = 0., sumZZ = 0.;

    for (int i = 0; i < length; i++)
    {
        sumXX += centerPCL[i].x * centerMCL[i].x;
        sumXY += centerPCL[i].x * centerMCL[i].y;
        sumXZ += centerPCL[i].x * centerMCL[i].z;

        sumYX += centerPCL[i].y * centerMCL[i].x;
        sumYY += centerPCL[i].y * centerMCL[i].y;
        sumYZ += centerPCL[i].y * centerMCL[i].z;

        sumZX += centerPCL[i].z * centerMCL[i].x;
        sumZY += centerPCL[i].z * centerMCL[i].y;
        sumZZ += centerPCL[i].z * centerMCL[i].z;
    }

    covariance.at<double>(0, 0) = sumXX / length;
    covariance.at<double>(0, 1) = sumXY / length;
    covariance.at<double>(0, 2) = sumXZ / length;

    covariance.at<double>(1, 0) = sumYX / length;
    covariance.at<double>(1, 1) = sumYY / length;
    covariance.at<double>(1, 2) = sumYZ / length;

    covariance.at<double>(2, 0) = sumZX / length;
    covariance.at<double>(2, 1) = sumZY / length;
    covariance.at<double>(2, 2) = sumZZ / length;

    return covariance;
}
```

Figure A 3. Example code of the Covariance Matrix Calculation

Transformation Matrix Code

```
/*******************************************************************
Function: CalculateTransformationMatrix()
Description : Calculates rigid transformations using SVD
Input : void
Returns : void
*******************************************************************/
void CalculateTransformationMatrix(Point3d P1, Point3d P2, Point3d P3, Point3d P4)
{
    //Variables for ICP algorithm
    Point3d RS_Points, Holo_Points;
    Mat R = Mat::zeros(3, 3, CV_64F);
    Mat T = Mat::zeros(3, 1, CV_64F);
    vector<Point3d> centerPCL, centerMCL;
    int vectorSize = 4;
    Mat covariance;

    vector<Point3d> realsense_3d = { {P1.x,P1.y,P1.z}, {P2.x,P2.y,P2.z}, {P3.x,P3.y,
{P4.x,P4.y,P4.z} }; // RealSense matrix - this will be obtained from calibration

    /*step 1 : find center of mass of both the datasets*/
    /*for RealSense*/
    for (int i = 0; i < realsense_3d.size(); i++)
    {
        RS_Points += realsense_3d[i];
    }
    RS_Points = RS_Points * (1.0 / realsense_3d.size());
    /*for HoloLens*/
    for (int i = 0; i < hololens_3d.size(); i++)
    {
        Holo_Points += hololens_3d[i];
    }
    Holo_Points = Holo_Points * (1.0 / hololens_3d.size());

    /*step 2 : centre the point cloud as per the center of mass calculated*/
    /*for RealSense*/
    for (int i = 0; i < realsense_3d.size(); i++)
    {
        Point3d pt;
        pt = realsense_3d[i] - RS_Points;
        centerPCL.emplace_back(pt);
    }
    /*for HoloLens*/
    for (int i = 0; i < hololens_3d.size(); i++)
    {
        Point3d pt;
        pt = hololens_3d[i] - Holo_Points;
        centerMCL.emplace_back(pt);
    }
    /*step 3 : calculate covariance matrix*/
    covariance = CalcualateICPCovarianceMtx(vectorSize, centerPCL, centerMCL);

    /*SVD registration method*/
    /*step 4: eigenvalues and eigenvectors from covariance matrix*/
    Mat w, u, vt;
    SVD::compute(covariance, w, u, vt, 0);

    /*step 5: calculate rotation matrix*/
    R = vt.t() * u.t();
    cout << "determinant of R: " << determinant(R) << endl;
    if (determinant(R) < 0.)
    {
        cout << "Reflection detected..." << endl;
        vt.at<double>(2, 0) *= -1.;
        vt.at<double>(2, 1) *= -1.;
        vt.at<double>(2, 2) *= -1.;
        R = vt.t() * u.t();
    }

    cout << "Rotation matrix calculated" << endl;
    for (int r = 0; r < R.rows; r++)
    {
        for (int c = 0; c < R.cols; c++)
        {
            cout << R.at<double>(r, c) << " ";
        }
        cout << endl;
    }
    /*calculate translation matrix*/
    Mat matPclCOM, matMclCOM;
    matPclCOM = T.clone();
    matMclCOM = T.clone();

    matPclCOM.at<double>(0, 0) = RS_Points.x;
    matPclCOM.at<double>(1, 0) = RS_Points.y;
    matPclCOM.at<double>(2, 0) = RS_Points.z;

    matMclCOM.at<double>(0, 0) = Holo_Points.x;
    matMclCOM.at<double>(1, 0) = Holo_Points.y;
    matMclCOM.at<double>(2, 0) = Holo_Points.z;

    T = matMclCOM - (R * matPclCOM);

    cout << "Translation matrix is: " << endl;
    for (int r = 0; r < T.rows; r++)
    {
        for (int c = 0; c < T.cols; c++)
        {
            cout << T.at<double>(r, c) << " ";
        }
        cout << endl;
    }

    Rotation = R.clone();
    Translation = T.clone();
}
```

Figure A 4. Example code of the Transformation Matrix Calculation

Iterative Closest Point Code

```
void IterativeClosestPoint(Point3d P1, Point3d P2, Point3d P3, Point3d P4)
{
	int iterations = 1;
	double oldError = 100;
	bool step = true;
	int maxIterations = 100;
	double minThreshold = 0.00001;

	while (step)
	{
		if ((iterations <= maxIterations) && !(abs(oldError - error1)
< minThreshold))
		{
			cout << "iteration number: " << iterations << endl;

			/*step 1 : Data transformation - from R and T matrix to
tranform RS coords close to HL coords*/
			CalculateTransformationMatrix(P1, P2, P3, P4);

			/*apply transformations to the datapoint*/
			for (int i = 0; i < realsense_3d.size(); i++)
			{
				Mat dataPt = Mat::zeros(3, 1, CV_64F);

				dataPt.at<double>(0, 0) = realsense_3d[i].x;
				dataPt.at<double>(1, 0) = realsense_3d[i].y;
				dataPt.at<double>(2, 0) = realsense_3d[i].z;

				dataPt = Rotation * dataPt;
				dataPt += Translation;

				realsense_3d[i].x = dataPt.at<double>(0, 0);
				realsense_3d[i].y = dataPt.at<double>(1, 0);
				realsense_3d[i].z = dataPt.at<double>(2, 0);
			}
			cout << "Difference in error: " << abs(oldError - error1)
<< endl;
			oldError = error1;

			/*calculate the error*/
			error1 = CalculateDistanceError();
			iterations++;
			step = true;
		}
		if (abs(oldError - error1) <= minThreshold)
		{
			cout
<< "************************************************************\n";
			cout << "Converged with error:" << abs(oldError - error1)
<< endl;
			step = false;
			break;
		}
		if (iterations > maxIterations)
		{
			cout
<< "************************************************************\n";
			cout << "Max iterations over, not converged" << endl;
			step = false;
			break;
		}
	}
}
```

Figure A 5. Code for the ICP algorithm