Imperial College London

IMPERIAL COLLEGE LONDON

DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING

Augmented Reality-assisted Human Robot Interaction

Student:
Aufar P. Laksana

CID:
01093575

Project Supervisor:
Professor Yiannis Demiris

Second Marker:
Dr. Tae-Kyun Kim

Final Year Project Report 2019

MEng Electronic and Information Engineering

June 19, 2019

Abstract

Powered wheelchairs are becoming increasingly commonplace in the modern world. However, a significant issue faced by powered wheelchair users (PWUs) is navigating the device in crowded areas. Controlling the powered wheelchair in crowded areas requires increased concentration from the PWU, as people in crowds often move unpredictably, or are hidden from view due to standing behind another person or object.

This project utilises computer vision techniques to predict the direction of travel of individuals in crowds and implements an augmented reality system using the Microsoft Hololens that helps the PWU by displaying visual aids that indicate the motion of people. The system further helps the user by warning them of potential collisions, allowing the PWU to make better navigation decisions. The project also explores the use of the system as a method of reactive control of the wheelchair, preventing collisions by stopping the powered wheelchair should the PWU not attempt to avoid a collision with an individual crossing their path.

The result of this project is an augmented reality system prototype that can be used to aid navigation tasks for PWUs. Our testing shows that the implemented system can detect collision risks in a controlled environment and react appropriately using only a visual camera input. This report also discusses the disadvantages of relying on only one form of sensor input for obstacle avoidance and the limitations of using the Hololens as a head mounted aid.

Acknowledgements

To Professor Yiannis Demiris, thank you for giving me the freedom to shape this project as I liked. You allowed me to explore fields that interested me and allowed me to work with the Hololens and robots that would normally be out of reach. This project has re-ignited my passion for engineering, robotics and making things work.

To the members of the Personal Robotics Lab, Rodrigo, Mark and Yong, thank you for always being willing to answer my questions, guiding me through the maze that is Unity, Hololens and ROS development. Without your help and support, this project would not have been possible.

To my friends and colleagues in the 5th floor 'office', Marek, Shrey, Tom, Zihan, Ian & Abdullah, thank you for sitting there with me as we worked on our separate final year projects. We have laughed, we have stressed, but ultimately we have all become friends. I will miss coming into the office every day and talking with you guys, and I wish you all the best of luck in whatever you choose to pursue.

To my Mum and Dad, Dian and Bawa, thank you for all the sacrifices you have made to allow me to study at this prestigious university. Dad, I have always seen you as a role-model to beat, and I have tried and will continue to try to surpass the expectations you have set. I will never be able to repay you for everything you have done for me, but I will continue to strive to make you both proud.

To my younger brother Ammar, who is about to start university. You have always supported me and been by my side. Despite our quarrels and disagreements, I could not ask for a better sibling. I hope you enjoy university and that you make the most of your time there.

Finally, to my girlfriend Emily, who despite the distance, has always been by my side these past four years. Thank you for all the support, the kind words, and the motivation to keep working hard. You have helped me grow as a person throughout my time at University, and I would not have been able to reach where I am now without you. I hope you end up deciding to do that PhD, you always were the smarter one.

Contents

Chapter 1

Introduction

1.1 Introduction

This report was written as part of the Final Year Project for the MEng Electronic & Information Engineering course. The project was supervised by Professor Yiannis Demiris at the Imperial College London.

1.2 Motivation & Objectives

Powered wheelchairs are becoming increasingly commonplace in the modern world. As technology advances, the devices have become more accessible and smarter, with schemes such as assistive control which help the user in navigation. Within the Personal Robotics Lab (PRL) at Imperial College London, one of the major research topics revolves around powered wheelchairs, and how the user experience can be enhanced. Previous work in the lab has utilized augmented reality headsets to enhance the user experience, by displaying visual aids to the powered wheelchair user (PWU) [1, 2] that help the PWU understand the internal state of the device, such as the current trajectory or planned path. However, despite the advances in smart wheelchairs and user enhancement techniques, a study showed that one of the concerns of PWUs is navigating in crowded spaces [3]. This concern is due to the unpredictability of people in crowds and their tendency to change direction rapidly.

As such, this project explores the development of an augmented reality experience using the Microsoft Hololens as an aid for PWUs. The goal of the system would be to detect people in the surroundings and display the corresponding visual aids to the PWU to indicate potential collisions. To further alleviate the concerns of PWUs, the system should also be able to avoid accidents by reacting appropriately when the PWU is unaware of a potential collision.

1.3 Challenges

This project highlights the problems associated with the current version of the Microsoft Hololens. First and foremost is the lack of a built-in library for accessing the raw frames

of the front facing camera, as well as a library for video streaming to partner devices. Secondly, the lack of a UWP implementation of the Robotic Operating System meant we were forced to modify a third-party library to be able to communicate between the Hololens running a Unity application and ROS nodes on the partner PC. Finally, we observed the limitations of the spatial mapping capabilities of the Hololens, and how they affect hologram stability and placement accuracy.

1.4 Contributions

By developing this augmented reality system, we have contributed:

- A Unity Engine application that can stream the front-facing camera of the Hololens across the network.
- A ROS package for the Darknet Neural Network framework.
- A trained object detector for detecting partially occluded people in crowds.
- A ROS wrapper for the OpenPose body pose estimation network.
- A Unity application that can visualize the position of detected persons by rendering holograms at the detected positions.
- A reactive control system for powered wheelchairs that avoid collision by manipulating the velocity of the device when it detects a collision risk.

1.5 Report Structure

This report begins with the background research that was done before embarking on the development of the system. We explore different computer vision techniques for people detection and tracking, as well as robotic mapping techniques for objects in the surroundings. We then discuss the requirements of the system we implemented, before describing the system from a high-level perspective, as well as why certain design decisions were taken.

The implementation section of this report covers the steps taken to develop the final system, from training the object detector to the reactive control system for the powered wheelchair. We then describe the testing procedure that allows us to analyse the performance of our product. Finally, we discuss the performance and limitations of the system we implemented before highlighting how the overall system could be improved in future work.

Chapter 2

Background

This project focuses on computer vision for detecting and tracking humans in the surroundings, estimating their trajectories and distance from the PWU, the reactive control systems that prevent collisions with the detected objects as well as the augmented reality display to provide visual cues to the PWU.

2.1 Human Detection

Human detection is a subset of the classic computer vision problem of object detection. To develop an augmented reality system that will help PWUs to navigate in public spaces, it is essential for the system to be able to discern humans from the surroundings.

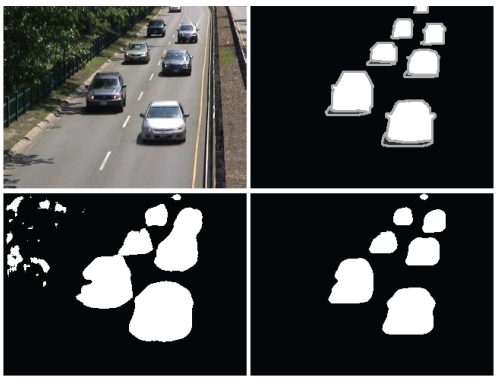
2.1.1 Direction of Research

The problem arises in crowded areas, whereby individuals are occluded by other people or objects in front of them, leaving only certain body parts visible. As such, we began our research with the problem of being able to detect people in images where identifying parts of the body are not always visible.

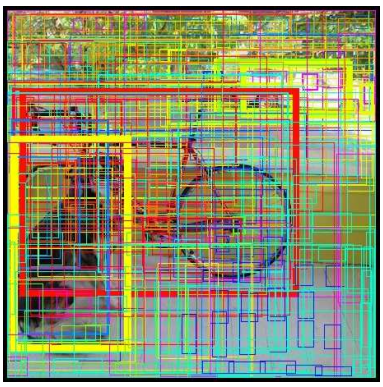
2.1.2 Review of Existing Methodologies

A related field of research is that of people counting and human detection in visual surveillance in public areas. Where the problem differs is that surveillance benefits from being able to rely on cameras with a good view of the crowd from above, whereas for a PWU, the camera will not have as high of a vantage point, making detecting each person in a crowd impossible.

Despite the disadvantage, similar techniques can be used to detect humans in video. Most methods are classified into two categories [4]. The first technique, foreground detection, attempts to model the background of an image and then detect the changes that occur between frames. The second category involves exhaustively searching the image with a scanning window and deciding if each window contains a human shape.



(a) Foreground Detection [5]



(b) Scanning Window [6]

Figure 2.1: Comparison of Foreground Detection and Scanning Windows. The number of bounding boxes drawn by the scanning window shows the computational complexity of the method.

Foreground Detection

Background subtraction is a widely used approach for detecting moving objects [7]. A temporal average filter can be used to find the median of all the pixels in an image to form a reference image. Frames with moving objects can then be compared pixelwise to the reference, and a threshold set to determines if the pixel is part of the background or foreground. People counting and human detection is achieved by segmenting the foreground image into individuals.

However, this technique often relies on a static camera in a well-placed location. This brings up several reasons as to why this method would not be suitable for this project. Firstly, the camera available is part of a head-mounted augmented reality device. The wearer can move the camera in 6 degrees of freedom. Secondly, the wearer will also be navigating a powered wheelchair. As a result, the background is constantly changing, and the reference image would require constant re-computation before human detection can even begin.

Scanning Windows

Due to the ever-changing surroundings of a mobile robot, a better approach for object detection is to exhaustively search an image using scanning windows and determining if an object is detected in each window. However, we note that this method is computationally expensive. To achieve real-time detection on a mobile robot, the use of a graphics processing unit (GPU) should be considered [8].

Classical Object Detection

Haar Cascades Haar cascades classify images based on the value of simple features [9], which are variants of the difference between the sum of pixel values in rectangular regions. An intermediate representation of the original image is used to compute a small set of representative rectangular features rapidly.

A cascade of classifiers is then used to determine if the region detects a human. The detection process is that of a degenerate decision tree, where a positive result in the first cascade will trigger an evaluation in the second, more successful classifier. As such, the initial classifier can eliminate a large number of negative examples with very little processing. After several stages, the number of sub-windows has reduced radically

Histograms of Oriented Gradients The method proposed is implemented by dividing the image window into small spatial regions and calculating a local 1-D histogram of gradient directions for all the pixels in the area. The combined local histograms form the overall feature representation of the image.

The detection window is tiled with the Histogram of Oriented Gradient (HOG) descriptors. In the original paper [10], these feature vectors were then used in a conventional SVM based window classifier to give human detections.

Deep Learning Object Detection

Modern approaches for human detection largely depend on Deep Convolutional Neural Networks (CNN). The method provides the best in class performance, as well as scaling effectively with more data. An added advantage of using CNN based object detection systems for this project is that they are also capable of detecting multiple classes of objects.

An issue with CNN approaches is that the methods are trying to draw bounding boxes around objects of interest in images. However, we do not know the number of objects in the image beforehand. As such, to be completely sure every object has been detected, a naive solution is to take a huge number of regions and attempt to classify all the objects in the region, a computationally expensive process.

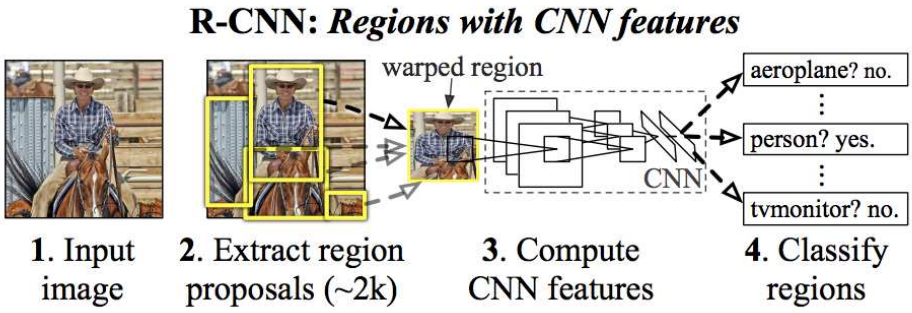


Figure 2.2: Visualization of the R-CNN approach. The number of region proposals contributes to the latency of this method.

R-CNN The R-CNN method uses a selective search to extract 2000 regions from an image [11]. The regions are selected by generating a large number of candidate regions

and using a greedy algorithm to combine similar regions into larger ones recursively. The regions are then fed into a CNN that acts as a feature extractor and the output dense layer consists of the features extracted from the image, which are then fed into an SVM to classify the presence of objects in the region.

The major disadvantage to this approach is the amount of time required to train the network. Each training image has to be classified once for each of the 2000 region proposals. Furthermore, the selective search algorithm is a fixed algorithm (no learning is done), and as such, could lead to the generation of bad candidate region proposals.

YOLO Whereas R-CNN uses regions to localize the object within an image, You Only Look Once (YOLO) looks at the image as a whole and uses a single CNN to predict the bounding box and the class probabilities [6]. By looking at the image as a whole, the network can use features from the entire image to predict each bounding box.

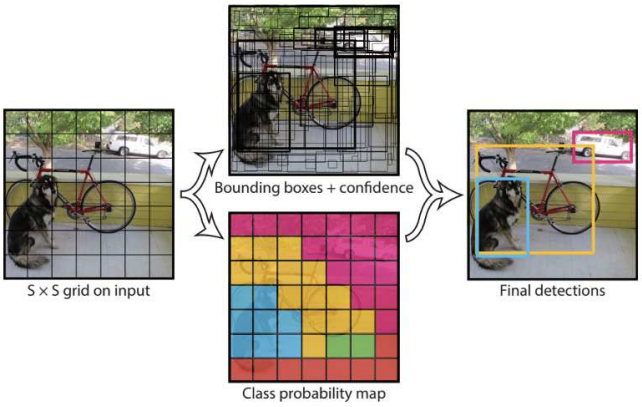


Figure 2.3: YOLO algorithm dividing a square image into grid-cells for bounding box prediction.

The model divides the image into an $S \times S$ grid, and for each cell, predicts a number of bounding boxes, the confidence for those boxes and the class probabilities.

2.1.3 Comments

As seen from the research, we can see that there are many ways to solve the human detection problem. The classical approaches, although computationally efficient, are significantly outperformed by the deep learning approaches. For a mobile robot in a public area, we want to be able to detect almost all humans in the surroundings to better inform the PWU.

However, the major disadvantage of the deep learning approach is the time taken to train the network, as well as the requirement of a GPU to achieve real-time performance. These issues will be addressed in a later section of the report.

2.2 Object Tracking

Object tracking can be defined as the ability to detect objects in consecutive frames and determining if the same objects are present. The techniques are often used in security and surveillance to track individuals across multiple cameras. A more relevant use of object tracking is in augmented reality with ARMarkers to allow for more accurate placements of holograms as the user moves through the AR world.

2.2.1 Direction of Research

A typical scenario for PWU in public spaces is having multiple people walking in the surroundings. Ideally, the augmented reality system should be able to track the same people across frames to be able to determine their direction of motion. As such, we focus our research on the multiple object tracking (MOT) problem in real-time. For an augmented reality system for a PWU, the object tracking must be done in real-time in order to provide feedback to the PWU. This narrows our field of research to online object tracking techniques.

2.2.2 Review of Existing Methodologies

Pedestrian detection is often achieved by using a high-quality object detector and associating the detections across frames [12]. The associations are based on the appearance and location similarity. Furthermore, it is possible to discern simple motion patterns from tracked pedestrians, allowing for more accurate tracking.

SORT

Methodology The Simple Online and Real-time Tracking (SORT) method relies on the accurate detections of a CNN to calculate bounding boxes of the tracked objects across frames [13]. The technique estimates the inter-frame displacements of each detected objects with a linear constant velocity model. The state of each tracked object is modeled using the bounding box centroids u and v, the scale and aspect ratio, s and r.

$$x = [u, v, s, r, \dot{u}, \dot{v}, \dot{s}]$$

When a new detection is associated with a tracked object, the bounding box of the new detection is used to update the tracked object state and using a Kalman filter to update the velocity components [14]. To determine associations between new detections and tracked targets, the SORT algorithm relies on the intersection-over-union (IOU) distance between each detection and the predicted bounding boxes of all the existing targets.

For every detection to be tracked, a unique tracker identity must be created and destroyed when the object enters and leaves the image. The original implementation of the algorithm relied on a IOU_{min} value to signify the existence of an untracked object. The tracks are then terminated if they are not detected for an allotted number of frames, to prevent the unbounded growth of trackers.

Limitations Due to the simplicity of the association metric, the significant overhead and complexity of object re-identification is removed, allowing for the system to work in real-time applications. However, this also reduces the accuracy of the tracking, since occlusions will spawn new trackers for the same objects. Furthermore, the accuracy of the tracking is largely dependent on the object detector providing accurate bounding boxes.

Deep SORT

The original SORT suffered from a high number of identity switches since the association metric was only accurate if the state estimation uncertainty was low. Wokje proposed a solution to the issue by learning a deep association metric on a re-identification dataset [15].

Methodology The tracking and Kalman filtering in Deep SORT is mostly identical to the original SORT implementation. However, Deep SORT uses a Mahalanobis distance as an association metric between the Kalman predicted states and new detections. It further uses a second metric, whereby an appearance descriptor is calculated for each bounding box. A gallery of the previous $L_k = 100$ descriptors is kept for each track. The algorithm then iterates and measures the smallest cosine distance between the existing tracks and the detection.

The appearance descriptor is implemented using a CNN that has been trained offline on a person re-identification dataset. The GitHub implementation of the Deep SORT algorithm uses a simple nearest neighbour query without any additional metric learning.

Limitations Although the accuracy of tracking is improved, and the issue of occlusions is reduced, the increased complexity of the algorithm requires more computational power. As stated in the paper, a modern GPU would be required to run this in real-time, due to the need for an appearance descriptor to be calculated for each detection.

2.2.3 Comments

For this project, we have limited ourselves to researching simple object tracking methods that work in real-time. We can see a trade-off between the accuracy of tracking and computational power. Further investigation into the hardware available and the importance of object tracker accuracy will be needed to decide what method would be best for the augmented reality system.

2.3 Head and Body Pose Estimation

Pose estimation is a general computer vision problem where we attempt to detect the position and orientation of an object. This process can be achieved by detecting key-

point locations that describe the pose of the object. For instance, in body pose estimation, we identify the joints in the body.

2.3.1 Direction of Research

An interesting concept to explore is that of head and body pose estimation as a way of inferring the direction a person is walking in. For instance, people tend to look in the direction they are currently walking, but should they want to change direction, they also tend to look in that direction before changing [16]. Similarly, if we can determine the body pose of a person, the system will be able to tell if a person is walking to or away from the PWU without relying on depth sensors.

2.3.2 Review of Existing Methodologies

Head Pose Estimation

Head pose estimation is intrinsically linked with visual gaze estimation [17]. If we can characterize the direction and focus of a person's eyes, it may be possible to determine the direction they will walk in next.

Facial Landmark Detection Before head pose estimation can be done, key-points on the face must be detected [18]. These points will then be used to solve a Perspective-n-Point (PnP) problem to determine the head pose. There are many facial landmark detection techniques, depending on the number of landmarks to be detected. As the number of landmarks increases, the more accurate the pose estimation can be. However, it also increases the complexity of the detection, and as such, it becomes a trade-off between the two factors.

Body Pose Estimation

An idea we wish to explore is using the body pose of an individual to estimate the direction they are walking in. If the system can discern between a person's back or front, we can infer the motion, since people do not normally walk backward. A limitation of our system is that it has to be done in real time for it to be effective. As such, the techniques we can explore are limited by the hardware available.

PoseNet A common approach for body pose estimation is to employ a person detector and perform single-person pose estimation for each detection, known as a top-down approach. PoseNet is a real-time human pose estimator with a web-browser implementation that runs on Tensorflow.js, making it easily available to anyone. The implementation is based on the works of Papanderou and Zhu [19] in building a network that utilizes the Faster-RCNN model as an object detector to obtain accurate bounding boxes of people in an image [20]. The key-points are then calculated using a a ResNet [21] by predicting activation heatmaps and offsets.

OpenPose Top-down approaches can be limited by the failure of the person detector. This is especially common when two individuals are very close to each other, and the detector is unable to differentiate between them. In contrast, the bottom-up approach, which is based on partitioning and labeling an initial pool of body part candidates into subsets [22], can deal with an unknown number of people and can infer that number by linking the part hypotheses.

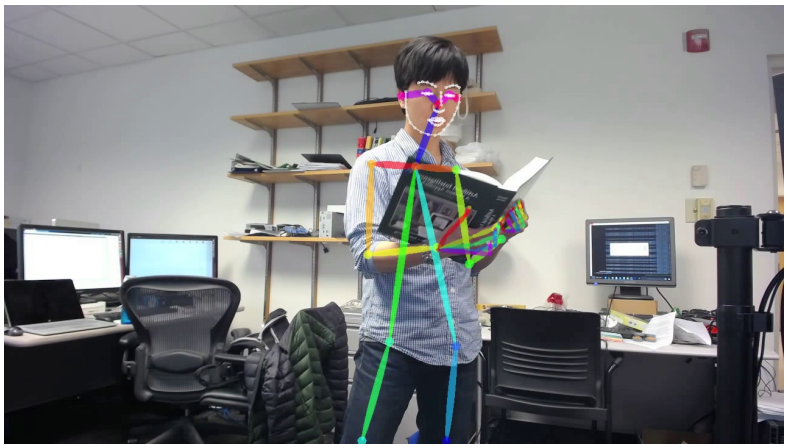


Figure 2.4: OpenPose body pose detection [23]. The network is able to determine an estimate of the key-points hidden by objects.

As such, OpenPose presents a method of multi-person pose estimation using a bottom-up approach [23]. The technique relies on *partial affinity fields* (PAFs), a representation that encodes unstructured pairwise relationships between body parts. The network produces the 2D confidence maps of body part locations and PAFs, and through greedy inference, the network outputs the 2D key-points for all people in the image.

2.3.3 Comments

Head and body pose estimation is a vast field of research, with dozens of effective real-time estimation methods. This project is not focused on achieving the best body/head pose estimation, but rather, in utilizing existing frameworks to infer directions of individuals. As such, we have refrained from delving too deep into the theory of body pose estimation, and instead, have attempted to choose a method from available implementations.

2.4 SLAM

The term mapping refers to a system that will create a map of the surrounding areas by detecting objects such as walls and other obstacles. To help users navigate, the system must analyse the surroundings for potential dangers. As such, it is important to build up a precise and complete map.

A fundamental method for robot navigation is the Simultaneous Localization And Mapping (SLAM) method. The process allows the system to predict the trajectory of the robot and the location of all objects online, without the need of an *a priori* knowledge of the robots location [24]. The method estimates the pose of the robot relative to landmarks which are detected. The popularity of SLAM increased with the emergence of indoor applications of robotic devices.

2.4.1 Direction of Research

For a PWU to navigate a wheelchair effectively through public spaces, they need to be able to avoid colliding with people or obstacles. An accurate map of the surroundings is key to solving this issue. However, some techniques rely on pre-existing maps of the area. A PWU may navigate their wheelchair to new locations, and can not rely on pre-existing maps for accurate navigation. Rather, the goal is to build up a real-time map of the surroundings that is accurate enough to avoid collisions.

2.4.2 Review of Existing Methodologies

A review of SLAM techniques can be found in [25], which also outlines the standard formulation of the SLAM problem as that of a Maximum a posteriori (MAP) estimation. The formulation relies on Bayes theorem and using the prior knowledge of the robots pose to maximize the likelihood to estimate the current position of the robot. The variables required to determine the position are the robot poses, the position of landmarks and the calibration parameters of the sensors.

To build an accurate map of the surroundings, the calibration of the sensors providing the measurements is a crucial step. The choice of sensors also matters, as the type of data returned by the sensor may affect the computational complexity of the SLAM algorithm. As such, it is common to have a module in the system that deals with the extraction of relevant features from the sensor data.

A fairly common assumption in SLAM approaches is that the world is static and remains unchanged as the robot moves. This becomes an issue with the goal of this project, which hopes to achieve the ability to detect humans walking around the wheelchair.

Visual SLAM

Visual SLAM (vSLAM) is an implementation of SLAM that relies on visual inputs only. As stated in [26], vSLAM is suitable for AR due to the low computational algorithms that can be implemented on the limited resources of an AR headset. The technique of vSlam is mainly composed of three modules:

Initialization In the initialization stage, camera pose estimation is conducted, to transform objects in a 2D image from the camera into a 3D coordinate system that the robot understands. This process determines the position and orientation of the camera relative to the object. A part of the environment is reconstructed as part of the initial map using the global coordinate system of the robot.

Tracking Here, the reconstructed map is used to estimate the pose of the camera with respect to the map. Feature mapping or tracking is conducted on the images in order to get a 2D-3D correspondence between the image and the map. The camera pose can then be calculated from the correspondences by solving the Perspective-n-Point problem [27]. This allows the system to identify where on the map the robot currently is.

Mapping When the robot passes through an environment that has previously not been mapped, the 3D structure of the surroundings is calculated from the camera images. The structures are then added to the existing map of the environment.

2.4.3 Comments

Due to the freedom in movement of an augmented reality headset camera, a system that relies solely on visual inputs may not be able to detect all obstacles in the surroundings. For instance, a limitation is that the PWU will not be able to extend their head backwards to view objects behind them. As such, it becomes important to consider the sensors available on powered wheelchairs, and utilize them to build an accurate map of the surroundings.

2.5 Augmented Reality Headsets

The improvements in augmented reality technology has spurred research into the use of AR devices in everyday tasks. The availability of commercial devices has also encouraged developments in the field, with products such as the Microsoft Hololens and the Magic Leap One.

2.5.1 Direction of Research

The augmented reality system built for this project needs to be able to give visual prompts to the PWU. As such, a device that already has the ability to create holograms is key. Furthermore, most AR devices have built in cameras to perceive the world around the user. We hope to be able to access the cameras on the device to do object detection and tracking.

2.5.2 Review of Existing Methods

Microsoft Hololens

The Microsoft Hololens is an untethered holographic computer, allowing for the display of 3D holograms pinned to real world objects. The Hololens is equipped with an array of sensors, making it an ideal choice of hardware for this project.

Holograms The Microsoft Hololens is able to blend real world and virtual content into environments where digital and physical objects can co-exist and interact. The term 'Mixed Reality' was first introduced by [28], and refers to the blending of the physical and virtual worlds.

The Hololens allows the developer to create 'Holograms', which are objects of light and sound that are displayed by the headset. Users are able to interact with the holograms through voice, gaze and gestures. Enhanced environment apps are applications that facilitate the placement of digital information on the user's current environment [29]. An example of an enhanced environment application is placing markers in augmented reality on objects that the user can interact with in both the physical and digital worlds.

Hardware Specifications As part of our research, we highlight the sensors on the device that may be relevant to the project. A full hardware specification is available online [30].

- 1 Inertial Measurement Unit (IMU)
- 4 Environment understanding cameras
- 1 Depth Camera
- 1 2MP Photo/HD video camera

Most importantly, the Hololens has a video camera. Preliminary research shows that it is possible to access the camera data directly, making it a suitable choice for the project.

Personal Robotics Lab The use of augmented reality devices to help PWUs is a research topic actively pursued by members of the Personal Robotics Lab at Imperial College London. Previous work has explored the use of augmented reality as a visualization tool to help PWUs understand the system dynamics of the wheelchair they operate, displaying visual cues that indicate the direction of travel of assistive control [1]. Other work involves using the camera to detect objects of interest in the environment, and developing a system that navigates the wheelchair to the detected objects through gaze and eye tracking [2].

2.5.3 Comments

Although other AR-devices exist on the market, due to the availability of the Microsoft Hololens in the Personal Robotics Lab, as well as the research done by individuals, it is in the best interests of this project to use the Hololens as the main augmented reality device for this project.

Chapter 3

Requirements Capture

3.1 Project Deliverable

The objective of this project is to develop an augmented reality system that can be used by powered wheelchair users (PWUs) to assist them in navigating their powered wheelchairs in public spaces with many people walking in the surroundings. The system should be able to detect the presence of individuals and infer their position relative to the PWU, and by extensions, estimate their direction of travel.

We propose a system that uses the Microsoft Hololens augmented reality headset as the primary input and visualization tool. The PWU would wear the headset as they operate the powered wheelchair, allowing the system to create visualizations of potential obstacles and collisions. Furthermore, the system would also encompass the reactive control aspect of the powered wheelchair. Should an individual be detected as walking in the wheelchairs trajectory, the system will send control signals to the powered wheelchair to slow down or completely stop depending on how far the target is from the wheelchair.

By definition of the requirements, we can divide the project into three parts: Human Detection and Direction, Obstacle Mapping & Visualization, and finally, Reactive Control.

3.2 Human Detection and Direction

The requirements of the Human Detection and Direction (HDD) system is to be able to use a video stream of the surroundings to determine the position and direction of people. The Hololens has a built-in camera that can be used to take photos of the surroundings of the user [2]. We will leverage this ability to create a video stream.

The actual HDD system is implemented on another computer with access to a GPU. We utilize the GPU to be able to do real-time object detection and pose estimation of detected individuals. The system should be able to infer the direction individuals are walking in, as well as their real-world positions relative to the PWU.

Features

- Creating a live video stream using the front-facing camera.
- Streaming the live video to accompanying computer.
- Object detector trained on humans/pedestrians.
- Object tracker to track detected humans, and determine their direction.
- Body/Head pose estimator to determine the direction of travel.
- Stream detections back to the Hololens for visualization.

3.3 Obstacle Mapping & Visualization

This project utilizes the Microsoft Hololens as a visualization and spatial mapping tool. The HDD system will output its detections and directions to the Hololens, which is used to create visualizations that will help the PWU. Examples of the visualizations include arrows that indicate the direction of movement, as well as alerting the user to potential collisions.

Features

- Receiving detection/direction data from HDD system.
- Utilize Camera to World transforms of the Hololens Camera to get World coordinates of people.
- Create holographic visualizations to help PWU understand the direction people are walking in.
- Create a map of obstacles for Reactive Control.

3.4 Reactive Control

The powered wheelchair (ARTA) available in the Personal Robotics Lab (PRL) can be manually operated using a joystick. The goal of the project is for the PWU to be able to wear the Hololens as an aid for navigation in public spaces. As such, it would be beneficial for the PWU if the wheelchair could reactively control the device to prevent collisions with detected objects.

Features

- Receiving object detections in front of the wheelchair.
- Prevent wheelchairs from driving into objects.

Chapter 4

Analysis and Design

This chapter gives an overview of the overall system and explains the design choices made. Throughout the project, we explored various methods to implement a real-time augmented reality system for PWUs operating a wheelchair. Naturally, the structure and goals of the project have developed since the interim report, and we review the differences between the initial goals and final product.

4.1 Design Overview

As stated in the requirements, this project consists of three major components:

- Human Detection and Direction (HDD)
- Object Mapping and Visualization
- Reactive Control on ARTA

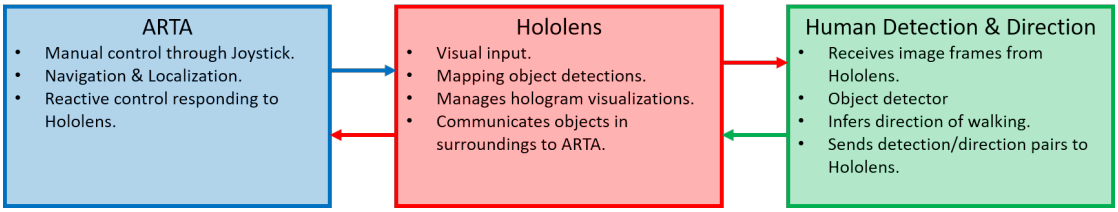


Figure 4.1: High level system diagram showing the flow of messages from ARTA and the HDD system through the intermediary device, the Hololens.

From a very high-level view, we can map these requirements to the respective devices they will be operating on. The HDD system implements the object detection and human direction inference, while the Hololens is responsible for utilizing the spatial mapping to obtain world positions of the detections, as well as visualizing the detections. The powered wheelchair (ARTA), has manual input that is overridden by the reactive control system that is dependent upon the detections and mappings. The diagram in Figure. 4.1 shows an overview of the system, and shows that the Hololens acts as an intermediary between ARTA and the HDD.

4.1.1 Hardware

	ARTA	Hololens	HDD
Hardware	Powered Wheelchair controlled by Laptop	Hololens	Desktop PC with GPU
Operating System	Ubuntu 16.04	UWP	Ubuntu 16.04

Table 4.1: Hardware description of devices each system runs on.

Table. 4.1 summarises the overall hardware the system is implemented on. The powered wheelchair, ARTA, is controlled by a laptop, which is responsible for the wheelchair speed, wheel rotations, navigation and localisation. The Hololens is a self-contained augmented reality headset, running the Universal Windows Platform (UWP) operating system. Finally, the Human Detection & Direction system is implemented on a desktop computer with a GTX 1050Ti GPU, allowing it to run real-time object detectors.

4.1.2 System Communication

Robotic Operating System

Since the project spans multiple operating systems, we have chosen to utilize the Robotic Operating System (ROS) as a means of communication between the devices. In ROS, a *node* is defined as a process that performs a computation. A node can be made up of smaller nodes that perform specific computations that serve the needs of the parent node. We can think of the three major systems as large ROS nodes that consist of smaller nodes that run individual tasks, such as creating the camera stream or detecting objects.

ROS Topics Nodes in ROS communicate with one another by publishing data in the form of *messages* which get broadcasted over a *topic*. Nodes can choose what data they receive by subscribing to topics. This method allows for nodes running on different devices to communicate with each other, regardless of the operating system. The nodes are unaware that the data it receives is published from a node running on a separate computer, making ROS a perfect choice for communication in this design.

4.2 Human Detection & Direction System

The HDD system is responsible for detecting and predicting the directions of people in the surroundings of the wheelchair. By taking visual inputs in the form of images from the Hololens, we run an object detector trained on a dataset of pedestrians to detect people and heads. The bounding boxes produced by the object detector are fed as inputs to an object tracker and a body pose estimator. We use the results of these two nodes to infer the direction a detected person is moving in, and publish the results back to the Hololens.

We present an overall view of the HDD System, covering the purpose and design of individual components. We also propose the reasoning behind certain design choices, which we cover in more depth later in this report.

4.2.1 YOLO Object Detector

Object detectors often form the input to an object tracker or pose estimation system [13, 31]. In the case of top-down body pose estimation methods, detections can be the first point of failure [32]. As such, the accuracy of the chosen object detector must be considered, together with the choice of using a pre-trained model or training on a more relevant dataset. Finally, we must also consider the use-case of the detector, which must be able to operate in real-time and detect moving objects as they pass by.

Choice of Detector

As commented on in Section 2.1.3, modern deep learning techniques outperform classical object detectors through increased accuracy but are limited by the requirement of a GPU to perform in real-time. Since the Hololens does not have built-in support to run object detection networks, Microsoft provides the Azure Cognitive Services API to allow developers to query their system for object detections. The limitation is that this service is not free and abstracts away the implementation of an object detector. Furthermore, one of the personal goals for this project was to learn more about CNNs in computer vision.

Taking this into account, we compared several deep learning architectures for object detection. Previous work done in the PRL used Facebook AI Research's (FAIR) Detectron to detect objects [2, 33, 34]. Further discussions with members of the Imperial Computer Vision & Learning Lab suggested the use of the YOLO object detector [6], due to its speed and having a lightweight implementation that can be run on lower-end GPUS at relatively high frame rates. This prompted the design decision to use the Darknet framework to use the **YOLOv3-tiny** architecture as the object detection method of choice for this project [35].

Pre-trained Model vs Training

An advantage of using the YOLO Darknet framework is that it provides trained models which can detect multiple object classes, including the class *Person*. One of the pre-trained models is the YOLOv3-tiny architecture trained on the Common Objects in Context (COCO) dataset [36].

Comparing Models To compare the accuracy of the bounding boxes produced by the pre-trained model, sample videos were recorded using the Hololens and used as a base comparison point. We observed that although the COCO trained model can detect individuals or multiple people who are well spaced out, it had difficulty in differentiating between people who are close together or slightly occluded. Figure 4.2 highlights the issue of the COCO model failing to detect small figures close together.

(a) COCO YOLOv3-tiny (b) Trained YOLOv3-tiny

Figure 4.2: Model comparison on a video from Sherfield Walkway at Imperial College London. The trained model is able to better detect figures at a distance.

Pedestrian Dataset A common issue in pedestrian detection is occlusion, where only certain parts of an individual are visible. To resolve this issue, we decided to train the YOLOv3-tiny model on the **CrowdHuman** dataset [37], which contains annotated images of people in crowded places. The annotations include bounding boxes for the head, visible human region, and full-body region. The annotations for partially visible people allows the network to learn to recognize occlusions, reducing the issue of failed detections when people are too close to each other. We also trained the network to detect heads, since we initially wanted to use head pose estimation to determine direction.

Analysis The result of training the YOLOv3-tiny model on the CrowdHuman dataset is that the system can better detect smaller figures with obscured bodyparts. The additional ability to detect heads allowed us to explore the use of head pose estimation for direction inference. We go into further detail on the training process in the Implementation section of this report.

4.2.2 YACHT: Yet Another Crowd Human Tracker

The bounding boxes produced by the object detector are consumed by the **Yet Another Crowd Human Tracker** (YACHT) module, which is made up of two nodes. The tracker node uses the Deep SORT algorithm to track detected individuals [15], while the pose estimator node uses the OpenPose [23] network to determine whether a person is walking towards or away from the PWU.

In the following sections, we briefly explain the methods used to infer the directions people are walking in. We also explore the use of head pose estimation and the limitations that prevented it from making it to the final design.

YACHT Tracker: Object Tracking

We explored existing object tracking methods in Section 2.2.2 and discussed our choices in 2.2.3. For moving object tracking on a powered wheelchair, we express the need for an online object tracking system. As such, we chose to investigate two related real-time methods, SORT [13] and Deep SORT [15].

SORT The Simple Online and Real-time Tracking (SORT) method is a fast online object tracker. The initial implementation of YACHT used the SORT algorithm due to its speed. However, it was quickly realized that due to the simplicity of the association metric, object tracking was not very accurate, especially for occluded objects. When two objects crossed paths, the tracker was unable to recognize the act and re-labeled the objects with new tracking IDs.

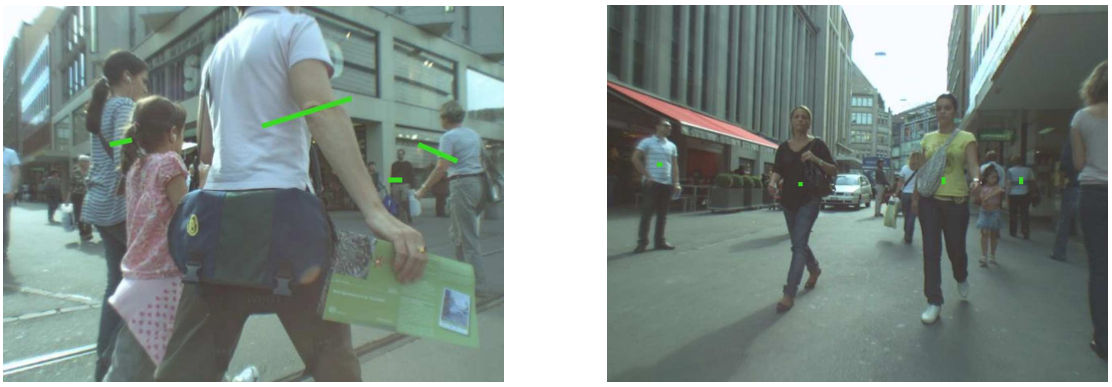
Deep SORT Deep SORT is an extension of the original SORT algorithm, but uses a deep network to generate feature descriptors for the predicted bounding boxes. We explain the algorithm in Section 2.2.2, but to repeat, instead of using the Intersection-over-Union association metric to compare bounding boxes, the deep network generates feature descriptors for the bounding box, and a Nearest-Neighbours is used to compare the features with a library of feature vectors for each tracked object. This is a form of person re-identification, and the additional step reduces the problem of trackers being lost due to occlusion.

Figure 4.3: MOT16 benchmark [38] (L) using our YOLO model (M) for Deep SORT (R). The increased detection accuracy results in slightly better initial detections, so tracking is more accurate.

Analysis The generation and storage of feature descriptors for tracks is an expensive process. For the system to run in real-time, a GPU is needed to accelerate the network. We have made changes to the Deep SORT implementation so it can run on Tensorflow-GPU, which we explain later in this report. This improves the speed significantly but uses up precious memory. As a result, we had to consider the amount of memory available on the GPU, since the YOLO detector and OpenPose networks also rely on GPU acceleration. After testing, we found that it was possible to run both networks on the GPU at the same time, and we chose to use the Deep Sort method.

Object Tracking for Direction Inference

An idea we explored was to use the previous image coordinates of a tracker to predict the direction a person will walk in. This involved storing the previous states of each track and extrapolating the centroids of each track to determine a direction.



(a) Tracks with horizontal motion

(b) Tracks with mostly vertical motion

Figure 4.4: Linear extrapolation works for objects that move across the frame, but it becomes difficult to determine the direction when mostly vertical motion occurs

Algorithm For each tracked object in a frame:

Data: (x,y) centroid image co-ordinates up to the previous 5 states
Result: (x,y) of extrapolated point
for *Frame* **do**
  for *Tracker* **do**
    if *Tracker existed in the previous frame* **then**
      Extrapolate over the centroids of previous states;
      Return linear extrapolation (x,y);
    end
    if *Tracker has no previous states* **then**
      Add (x,y) centroid of tracker to queue of previous states;
      Return current centroid (x,y);
    end
  end
end

Issues Although the method works for trackers which cover relatively large distances across the frame, linear extrapolation of image co-ordinates suffers when the tracked centroid does not have much horizontal motion. As such, this leads to an ambiguous definition of the direction. Since the object is not moving across the screen, it is not possible to differentiate between a person standing still, moving directly towards the PWU or walking away.

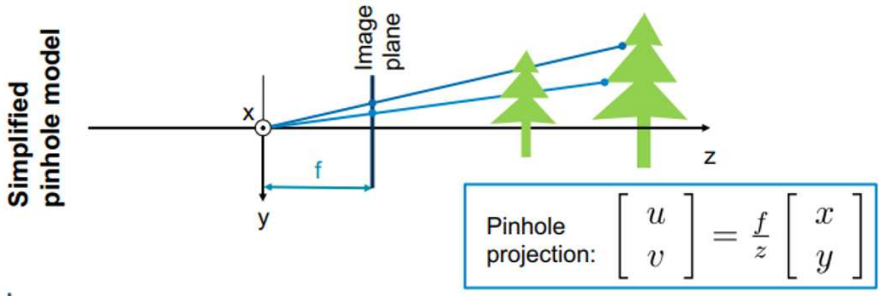


Figure 4.5: Pinhole Camera projection and the loss of the z-axis [39]. As such, it is not possible to determine the exact distance to an object without a depth camera.

In the pinhole camera model when an object in the real world is projected onto a 2D image, we lose the distance along the z-axis. The issue of a world-to-camera image projection occurs when we want to obtain the world co-ordinate of an object from the image. Due to the loss of z-axis information, the best we can do is to calculate a ray through the 2D image point. However, as shown in Figure 4.5, it becomes impossible to tell how far away the object is without a depth camera since the object can exist anywhere along that ray.

YACHT Direction: Body Pose Estimation

To solve the direction ambiguity brought up in Section 4.2.2, we proposed the use of body pose estimation techniques to determine whether a person is walking towards or away from the camera of the PWU. We researched several body pose implementations in Section 2.3.2, but we ultimately decided on OpenPose, due to its well documented implementation on GitHub [23].

Object Detectors & Bottom-Up Approaches The YOLO object detector outputs the original image and the associated bounding box coordinates of detections. The OpenPose node consumes the image and performs body pose estimation on the whole image, before matching poses with the object detections. Since OpenPose is a bottom-up approach, we admit that it is counter-intuitive to use an object detector to detect individual people when OpenPose determines the body part key-points across the whole image. This will be explored more in the evaluation of the report.

Keypoint Estimation The OpenPose framework provides several pre-trained models for body pose keypoint estimation. From our tests, we found that the *BODY_25* model was the fastest, suiting our real-time requirements. Figure 4.6 shows the key-points generated by the model. The model identifies 25 key-points on the human body and can differentiate between the left and right limbs on the human body. This makes the model suitable for determining if a person is facing the camera or not. We further explain the methodology in the implementation section of the report. From this, the node outputs the direction of the object to the Hololens.

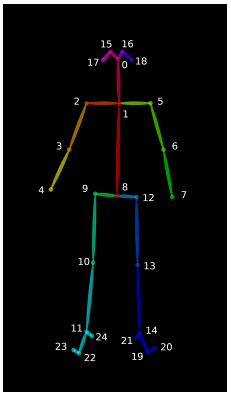


Figure 4.6: Keypoints produced by the BODY_25 model [23].

Head Pose Estimation

We initially began the project by exploring the use of head-gaze estimation as a novel way of inferring the intended direction of motion of a person. We researched the concept of head pose estimation in Section 2.3.2, with the logic being people tend to look in the direction where they are walking. We leveraged the use of the DeepGaze library as an initial starting point [40] since the library has a built-in head-pose estimator.

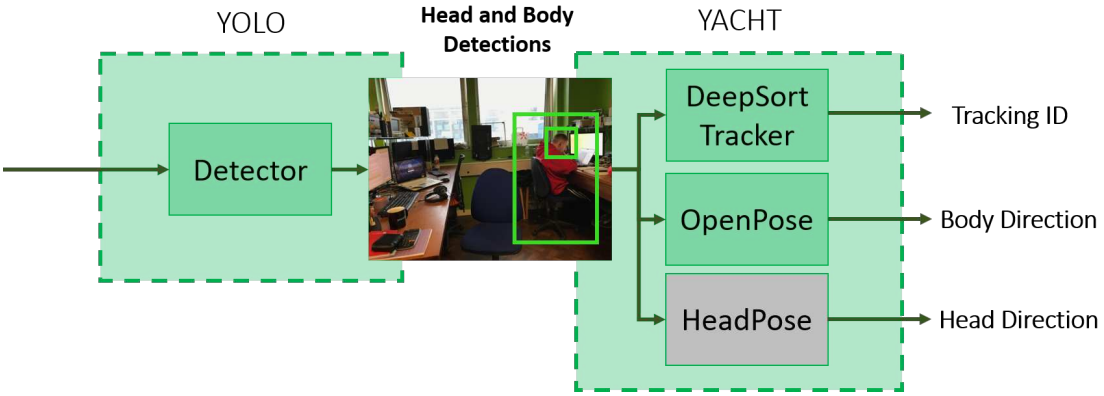
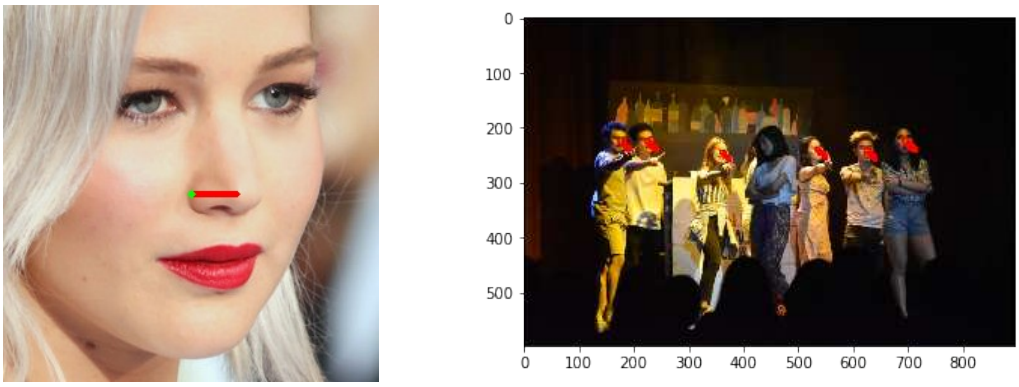


Figure 4.7: Initial HDD System with the HeadPose Node before removal.

Head Detection For this approach, we trained the YOLO detector to detect heads using the annotated CrowdHuman dataset. The head detections are consumed by the HeadPose node, which produces head pose projections that are sent to the Hololens.

Reasons for removal We noticed from our research that the head pose estimation was not very accurate, as can be seen in Figure 4.8. A close up image of a face still returns an inaccurate estimate of the head pose. For smaller faces with multiple detections, the head pose estimation was not performed in real-time and also produced incorrect estimates. Finally, as we will explain in the implementation section, the quality of the images received from the Hololens was too low, and as such, facial landmark detectors required for head pose estimation were unable to detect the key-points.



(a) Head pose estimation on close-up of face. (b) Estimation on people at a distance.

Figure 4.8: DeepGaze head pose estimator on sample images. We notice in (b) that DeepGaze predicts everyone to be looking in the same direction, despite the differences in head poses.

4.3 Hololens Unity Application

The Microsoft Hololens is a key component of this project since it acts as the main input, visualization, and mapping device. To begin, we utilize the world-facing camera, which sees what the PWU is looking at as the input to the HDD system. Further along the processing pipeline, the Hololens Unity application receives the image coordinates of human detections and respective directions to build up a map of the objects in the surroundings. The application manages the holograms, keeping track of the world coordinates of the objects, which it sends to ARTA for the reactive control of the wheelchair. This section covers how the system is designed so that it can be used in conjunction with other ROS nodes despite being a Unity application. It also describes the approach used to develop a front-camera stream, as well as a high-level description of the modules responsible for the world mapping and hologram visualization.

4.3.1 ROS Node

Both ARTA and the HDD are implemented on the Robotic Operating System as ROS nodes and communicate with their sub-nodes using ROS messages across topics. As explained in Section 4.1.2, we view each of the three sub-systems as individual ROS nodes. However, the Hololens does not natively support ROS.

Since ROS is unable to be run on UWP, we had to use the **ROS#** library implemented by Siemens. The library allows Unity applications to send and receive data in the form of ROS messages across topics. Figure 4.9 shows how the Unity application is viewed from other ROS nodes as just another node which it can communicate with using topics. The full implementation details of the ROS wrapping are covered later in this report.

4.3.2 HoloCamera

We begin the Unity application analysis by starting at the beginning of the system pipeline. The HoloCamera is the main visual input to the whole system, prompting

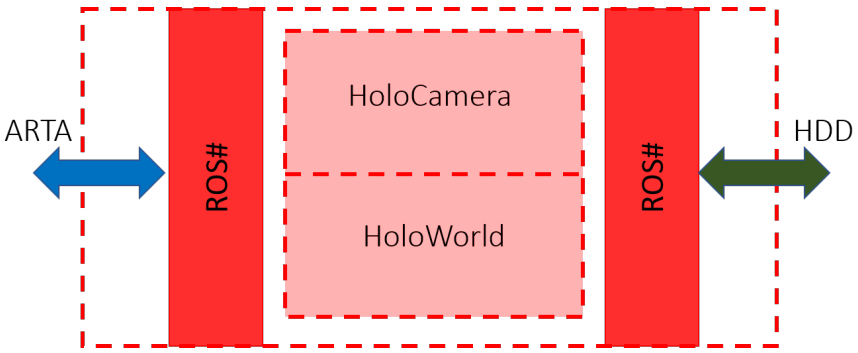


Figure 4.9: ROS# acts as a ROS wrapper around the Unity application, allowing for seamless communication with other ROS nodes.

the development of a video camera stream. Previous work done in the PRL have used stationary images taken with the Hololens, but the process of streaming live video from the front camera is a new direction of development. Due to the importance of this step, we spent time researching methods of video streaming to an external computer.

Video Streaming Choices

Windows Device Portal Microsoft provides a portal to access the Hololens device configuration from a web browser. From the portal, we can manage the connection, control what applications are running and most importantly, obtain a video stream of the front-facing camera. However, after some testing, we realized that there is a delay of 1-2 seconds between the camera and the video rendered on the computer, making it unsuitable for this project.

Microsoft HoloLensForCV Another option that was explored was the computer vision development tools released by Microsoft for the Hololens. This library provides developer access to the live camera stream, as well as the raw sensor data, such as the depth and IMU. This would have been the ideal video streaming choice. However, at the beginning of the project, we were not so experienced with developing UWP applications in C#. Furthermore, it proved challenging to stream the sensor and video streams from a UWP device to a Linux machine, since Windows and Linux have different data formats and standards.

Unity Camera Stream From our research, we found the **Vulcan Technologies Hololens Camera Stream** Unity add-on library. The community support for Unity development on the Hololens is immense, with various libraries such as the Mixed Reality Toolkit (MRTK) and many other Hololens specific tools. Furthermore, the 3D world modelling and spatial mapping capabilities available in Unity extend the capabilities of an augmented reality system. It abstracts away the complexities of the raw Hololens sensor data, reducing the time to market of applications. Finally, previous work in the PRL involving the Hololens has relied on Unity, making it an ideal choice for this project.

Module Description

The HoloCamera module is responsible for accessing the front-camera of the Hololens, compressing the raw image data into a JPEG format and streaming the video frames over the network. The application produces a ROS Compressed Image message that is sent to the HDD for object detection. We provide a complete explanation of the process in the implementation part of this report.

4.3.3 HoloWorld

The goal of this project was to develop an augmented reality system that will assist PWUs in navigation by providing visual cues of people in the surroundings. Unity applications can place holograms in the user's surroundings, and render them on the Hololens screen for the user to see. While the majority of the object detection and inference is done on an external computer, the Unity application on the Hololens is used to convert the image coordinates of objects into their 3D position in the world. Furthermore, the map of the surroundings is used by the reactive control component of ARTA to avoid collisions with the detected object, making the HoloWorld module a key component of the overall system.

World Manager

Since the Hololens is the intermediary, the World Manager sub-module is responsible for managing all the ROS topics between the Hololens, the HDD system, and ARTA. The module receives the image coordinates from the HDD system and projects the detected points into the world frame. This returns a set of world coordinates which we can assign holographic visualizations and map in the Unity world surrounding the Hololens.

ARTA Manager

This module is a representation of ARTA in the Hololens world. It subscribes to topics published by ARTA that contain information about the robot, such as the position in the real world, the linear velocities, and other published topics.

Alignment With most visual SLAM implementations, the camera attached to the mobile robot is fixed to the frame of the robot. This simplifies the conversion of the image co-ordinates to the robot frame, and then to the world frame. However, when a PWU wears the Hololens, the front-camera can move in 6 degrees of freedom. Most importantly, the PWU can turn their head to look left and right. This brings up the issue of not knowing whether a detected object is actually in front of the powered wheelchair, or if the PWU is looking at an object to the side.

We visualize this problem in Figure 4.10. When the Hololens camera and wheelchair frames are aligned, it is easy to tell if an object is in the path of the wheelchair. For the reactive control component of ARTA, we need to know if an object is in ARTAs current

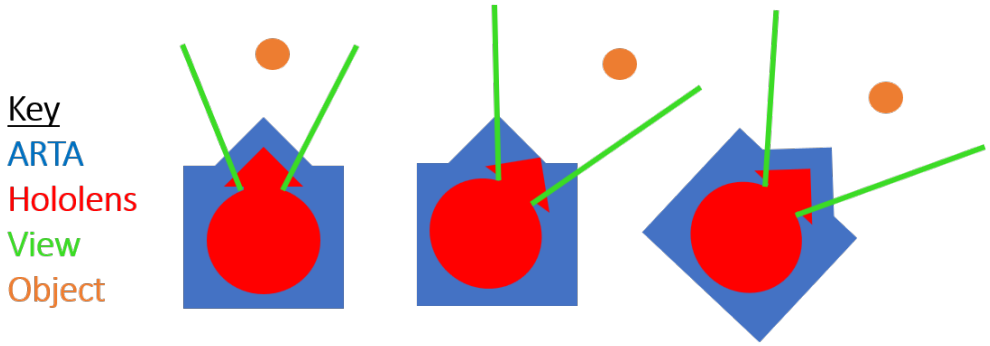


Figure 4.10: We need a way of knowing if a detected object is in front of the wheelchair, or if the PWU is looking to the side.

trajectory to determine if a collision will occur. As such, This module is responsible for discerning between people being in front of the wheelchair or to the side.

Reactive Control Furthermore, the ARTA Manager is responsible for consuming the navigation and localisation topics from the powered wheelchair and translating them to positions in the AR world of the Unity application. By modelling the wheelchair as an object in this world, the reactive control system can calculate the distance between the wheelchair and people detected in the surroundings. We can convert these distances and combine them with the manual input signals received from ARTA to avoid collisions with the detected individuals.

4.4 ARTA

The Personal Robotics Lab (PRL) at Imperial College London have built and developed an original smart powered wheelchair known as the Assistive Robotic Transport for Adults (ARTA). The wheelchair was designed to assist adults with mobility issues and is frequently used in the PRL as a research topic in conjunction with exotic control interfaces such as the Hololens [1, 2].

4.4.1 ROS Packages

Due to the previous work done on ARTA, the PRL have developed several ROS packages for the control of the powered wheelchair. These packages are used to combine data from the sensors attached to the wheelchair, using the data for localisation of the mobile robot and accepting manual input from the PWU in the form of joystick commands. We use these packages as a base system to build upon for our reactive control module.

4.4.2 Reactive Control

The powered wheelchair implements the physical part of the reactive control system project goals outlined in Section 3.4. By understanding the PWU input control signals from the joystick of the wheelchair, a system was developed which uses the signals

Figure 4.11: A PWU wearing the Hololens controlling ARTA using the joystick.

to determine if collisions with people in the surroundings will occur. The system also implements the reactive control aspect, which overrides the control signals sent by the PWU to avoid collisions.

4.4.3 Removal of Exotic Control Interfaces

In the interim report, we proposed the use of eye gaze tracking as a form of wheelchair control. We proposed the use of the Pupil Labs eye tracker add-on for the Hololens as a way of determining a point the user is looking at and moving the wheelchair autonomously there. We further proposed the use of the eye tracker to determine whether the PWU had noticed a person walking into the trajectory of the wheelchair. Fortunately, the members of the PRL have already implemented similar control interfaces for ARTA using the Hololens, and developed them in the `wheelchair_control` ROS package.

As the project progressed, we encountered an issue with image compression. Due to the limitations of the Unity engine, image compression of frames captured by the front-facing camera can only be done on the main thread of the application. This thread is also responsible for displaying the augmented reality experience and rendering holograms. As such, the Unity application displays its content at 5 frames per second (FPS), well below the targeted 60 FPS level recommended by Pupil Labs for augmented reality experiences. We explain this issue in the implementation section of the report. Through experimentation, we found it was difficult to use the eye tracker plugin provided by Pupil Labs and maintain a video stream of the front-facing camera simultaneously. Code profiling showed that the main thread spent more time compressing the raw images into JPEG format, and this did not give ample thread time for the eye tracker plugin to work properly. As such, we decided to remove the exotic control interface to focus on the reactive control aspect of the project.

Chapter 5

Implementation

This chapter is concerned with the implementation details of the individual components introduced in Chapter 4. This includes the development of the Unity application responsible for producing the front-facing camera video stream and displaying the visual cues, the development of the human detection and direction system, as well as the reactive control systems implemented for the control of ARTA. Previous work in the PRL had utilized the Hololens camera to capture images that were then processed on an external computer, but where this project differs is that a video stream is required to perform real-time object detection. As such, a large amount of time was spent at the very beginning of the project trying to produce a video stream, since the whole project depended on this form of visual input.

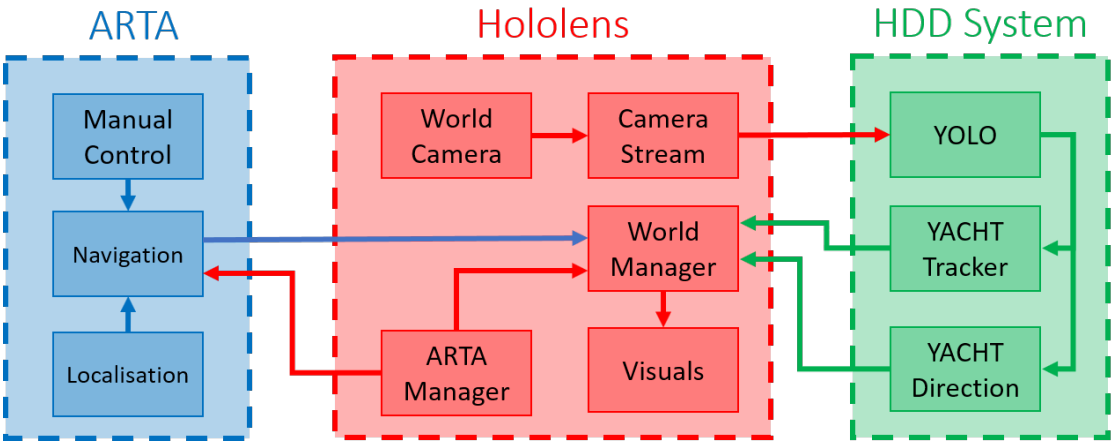


Figure 5.1: System diagram detailing individual components of the programs running on each device.

Figure 5.1 is a more detailed diagram of the high level system digram presented in Figure 4.1. We show the communication between the three separate devices, and how each node can be broken down into smaller nodes running specific computations. For the rest of this report, we represent the ARTA, Hololens, and HDD system components with the colours blue, red, and green respectively.

5.1 Human Detection & Direction System

In order to determine the direction people are walking in, it is necessary for the system to be able to detect humans. Only after detection is it possible to discern the motion of individuals, which can be achieved through object tracking and body pose estimation. Figure 5.2 shows the breakdown of the HDD node into components responsible for these two tasks. This section is concerned with the implementation of the methods needed to perform the direction prediction, as well as how the system communicates between its nodes.

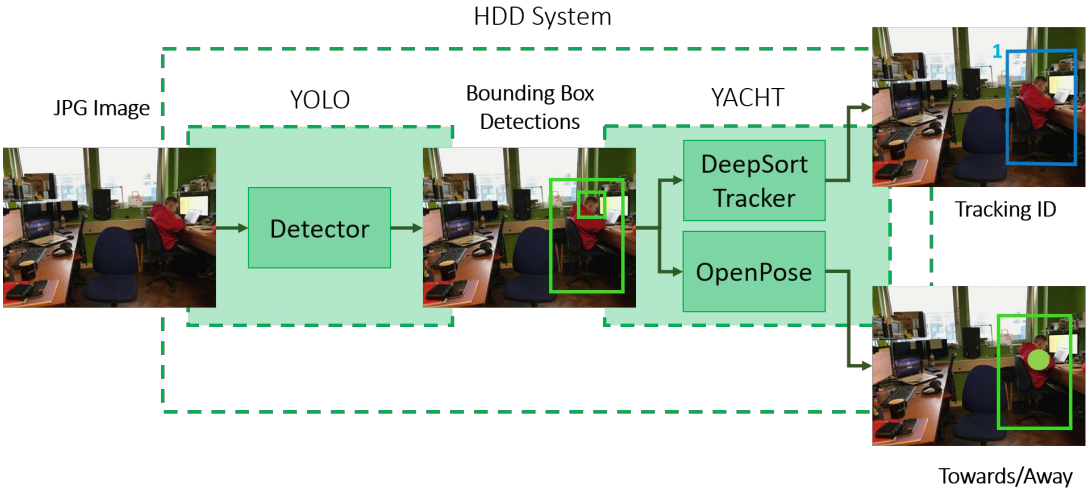


Figure 5.2: The HDD System is sub-divided into two ROS packages, YOLO and YACHT. We show that YACHT has two sub-nodes which both depend on the outputs of the detector.

We begin this section by listing the hardware requirements for the system. Due to the nature of the HDD, and its reliance on modern deep learning techniques, access to a modern GPU is essential. We refrain from trying to explain the step-by-step process to setup the hardware and software, and instead, point the reader in the direction of an article[1] which covers this topic.

5.1.1 Hardware & Software Dependencies

Hardware

We implemented the HDD system on a desktop computer connected to the Imperial College network. Due to the real-time computer vision requirements of this project, the computer was chosen due to the GTX 1050Ti GPU with 4GB video RAM available on the system. The computer was also equipped with an Intel i7-2600 CPU and 8GB DDR3 RAM.

[1] https://link.medium.com/xQ5w2FMXoX

Software

We have refrained from posting all Python dependencies for the project and only mention the important ones. A complete listing is available in the corresponding repositories of the packages. The following software dependencies are required to run this project:

- Ubuntu 16.04
- Python 2.7
- OpenCV 3.0
- ROS Kinetic

Due to the deep learning component of the project, the following software dependencies are essential:

- Nvidia Graphics Drivers
- CUDA 8.0 Toolkit
- cuDNN 6.0
- Darknet
- Tensorflow-GPU
- Caffe

5.1.2 YOLO Object Detector

As mentioned in Section 4.2.1, we chose to use the YOLOv3 Tiny architecture and trained it on the CrowdHuman dataset. We begin this section by introducing the reader to **Darknet**, the neural network framework YOLO is implemented on, and how we integrated it into ROS. We also briefly explain how the network detects objects and compare YOLOv3 Tiny with the more memory intensive YOLOv3. We then guide the reader through the training process, and the analysis we did to validate the human detection improvements compared to pre-trained models.

Darknet

Darknet[2] is an open source neural network framework written in C and CUDA which supports both CPU and GPU computation [41]. The source code for the framework is freely available on GitHub, and it can be used to train different neural network architectures like more conventional deep learning frameworks such as Tensorflow or Caffe.

[2]https://pjreddie.com/darknet/

Darknet in ROS

By definition, ROS is language-independent, although, at the time of writing, three main libraries have been defined for ROS, making it possible to program ROS in Python, Lisp or C++. On the other hand, Darknet is implemented in C, due to the speed of compiled low-level languages in conjunction with CUDA. However, the Darknet framework is compiled into *Shared Object (.so)* file, which is analogous to a Windows DLL. As such, it becomes possible to access the framework by writing wrappers around the compiled library file.

Darknet has basic Python wrappers around the compiled library which convert Python data types into C and vice versa. However, the original wrappers for detection are written to run on images that are saved on disk. Darknet converts the saved images into a C data structure `IMAGE` and performs the detections. To integrate the framework into ROS, the node must be able to receive data from image topics with `CompressedImage` or raw `Image` messages.

In ROS Python, the JPG or PNG images received from the `CompressedImage` message can be converted to numpy arrays which store the RGB values, without the need to be saved on disk. As such, we wrote Python wrappers for Darknet that allow the framework to support images in the form of numpy arrays as well as images saved on disk. The following listing defines the `IMAGE` data structure, and the conversion of an image numpy array to the Darknet format:

```
1  # IMAGE: a C data structure used by Darknet
2  class IMAGE(Structure):
3      _fields_ = [("w", c_int),
4                  ("h", c_int),
5                  ("c", c_int),
6                  ("data", POINTER(c_float))]
7
8  # Converts numpy array to Darknet IMAGE type
9  def nparray_to_image(img):
10     data = img.ctypes.data_as(POINTER(c_ubyte))
11     image = ndarray_image(data, img.ctypes.shape,
       img.ctypes.strides)
12
13     return image
```

Listing 5.1: Darknet IMAGE Python wrappers for seamless ROS integration.

Further Python wrappers were written for the detection and return of the image bounding box co-ordinates. We also created ROS messages for the bounding box detections, which allows the Darknet YOLO node to communicate with other nodes in the HDD system. The full code listing can be found in the Appendix [3,4].

[3]https://github.com/alaksana96/darknet-crowdhuman
[4]https://github.com/alaksana96/fyp_yolo

YOLO Detection Algorithm

As researched in Section 2.1.2, the algorithm divides an input image into a $S \times S$ grid. Each grid cell can predict one object, and a cell can predict a fixed number of bounding boxes B, which we visualize in Figure 5.3. The predicted bounding box is chosen from the set of B boxes with the highest box confidence score, which is a measure of how likely the box contains an object and how accurate the boundary box is. It also predicts the conditional class probability, which is the probability a detected object belongs to a particular class.

(a) A grid cell can make B predictions, in this example $B = 2$.

(b) The bounding box with higher box confidence score is used.

Figure 5.3: Visualization of the YOLO person detection algorithm dividing a resized square image into grid cells.

YOLOv3 Tiny vs YOLOv3

While testing out the different models, we noticed that the YOLOv3 model was consistently crashing and causing segmentation faults. On further investigation, we noticed that this was due to the network using up all 4GB of video memory available on the GPU. In comparison to its predecessors YOLO and YOLOv2, YOLOv3 is a much larger network which has 106 fully convolutional layers. Although it is far more accurate at predicting bounding boxes, it reduces the frame rates that can be achieved on video. As such, we decided to use YOLOv3 Tiny, a shallower variant of the network that is suitable for real-time image detection. Although the tiny version is not as accurate, it is much lighter on memory, using less than 1GB video RAM, making it a suitable choice for this project.

Training YOLOv3 Tiny

CrowdHuman The CrowdHuman dataset [37] is a benchmark dataset to evaluate detectors in crowd scenarios better. The essential features of the dataset are the size, qual-

ity of the annotations, and diversity. Each image contains multiple people with varying degrees of occlusion, which allows for object detectors to better learn the representation of obscured people.

	Caltech	KITTI	CityPersons	COCOPersons	CrowdHuman
# images	42,782	3,712	2,975	**64,115**	15,000
# persons	13,674	2,322	19,238	257,252	**339,565**
# ignore regions	50,363	45	6,768	5,206	**99,227**
# person/image	0.32	0.63	6.47	4.01	**22.64**
# unique persons	1,273	$< 2,322$	19,238	257,252	**339,565**

Figure 5.4: A comparison of CrowdHuman to other person image datasets [37], showing the increase in number of people and diversity in the images.

As can be seen from Figure 5.4, the dataset contains far more unique identities. By examining the dataset, we can see that it contains people in a wide array of situations, at varying distances with different body poses.

Annotations The CrowdHuman dataset provides its annotations in the `.odtg` format, which is a variant of JSON. Each line in the annotation file corresponds to a JSON containing the image ID and the bounding boxes. The annotations include boxes for the *visible box*, *full body box* and *head box*. For training, we chose to use the full body and head boxes only. We wanted the detector to be able to learn to predict occluded individuals, and we also wanted to experiment with head pose estimation. Each bounding box is annotated as below, where `x,y` is the top-left corner of the bounding box. The `width` and `height` are also given in image co-ordinate pixels.

```
[x, y, width, height]
```

On the other hand, to train YOLO on Darknet, the annotations must be given in a completely different format. The Darknet annotation format is as such:

```
<object-class> <x> <y> <width> <height>
```

Since Darknet accepts images of any size; it works with image units which are scaled relative to the size of the image. As such, all the values for the bounding boxes are between 0 and 1. Furthermore, the `x, y` values in the annotation are measured from the centre of the bounding box.

Converting Annotations To use the CrowdHuman images as a training set for the YOLOv3-tiny model, we had to write several scripts that converted the annotations to the Darknet format. We have included these scripts in the Appendix should the reader wish to convert the dataset themselves[5].

[5]https://github.com/alaksana96/darknet-crowdhuman/blob/master/README.md

Training Parameters Before training, we set up the model configuration file to learn two classes, head, and body, as well as to use batches of 32 divided into a subdivision of 8 images. This limits the number of images loaded into memory at once to 8, to prevent running out of GPU memory. We also reduce the size of the training images to 416×416 pixels, to further reduce the GPU usage.

```
batch=32 %Training parameters for YOLO Tiny
subdivisions=8
width=416
height=416
channels=3
momentum=0.9
decay=0.0005
```

Listing 5.2: Training parameters used for YOLOv3 Tiny on the CrowdHuman dataset

These optimizations were done to maximize the amount of GPU memory used for training, without a sudden surge in usage causing a segmentation fault. Resizing the input training images allows the algorithm to divide the image into a $S \times S$ grid. Generally, the larger the height and width, the better the predictions, since the image can be divided into more grid cells. This is a trade-off we had to make to be able to train the network on a mid-range GPU.

Training Process We left the model to train overnight, creating backups of the weights every 1000 iterations. The following day, after reaching 30,000 iterations, we decided to stop training the network. The average loss error and total loss had stopped decreasing for several hours and was hovering at around 29.134. We reasoned that the network had reached a minimum, and further training would be redundant since it would over-fit the dataset. The final weights are available in the file `yolov3-tiny-crowdhuman.backup`.

Evaluating Trained Model

Hololens Videos As mentioned in Section 2.1.2, we recorded several test videos using the Microsoft Hololens. These videos were capture in various locations on the Imperial College campus. Figure 5.5 shows the model detecting people at range.

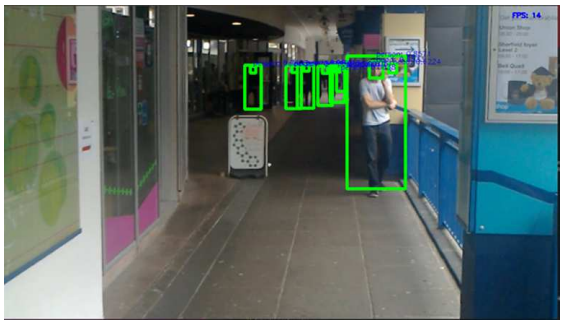


Figure 5.5: Trained model detects people and heads at different ranges. We also see it can detect people far away and accurately detect their heads.

A common area where lots of people frequent is Sherfield walkway. As seen in Figure 5.6, this was an ideal place to capture a video since it features a lot of people walking around in different directions. We obtained several more videos outside the EEE building and inside the 5th floor ICRS lab.

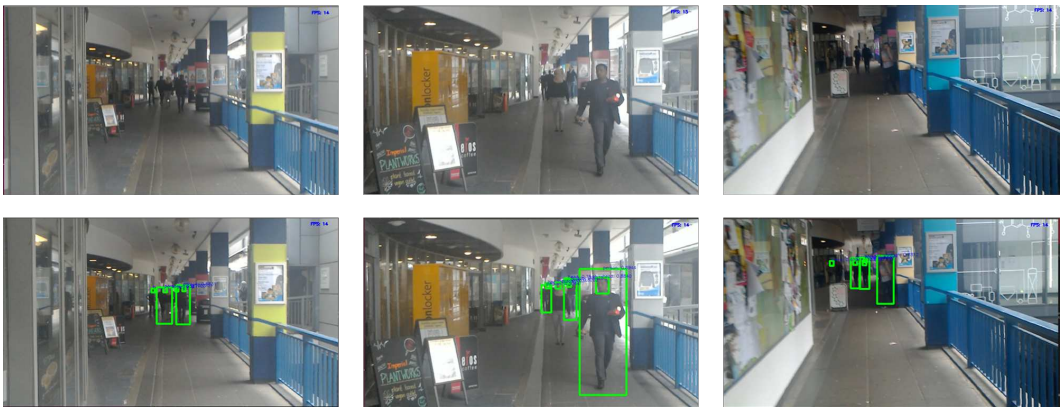


Figure 5.6: Evaluating the model on Sherfield Walkway test video. We notice that it is able to detect small figures close together at a distance.

The videos from the Hololens were captured whilst a person was walking with the device, to best emulate a PWU in a wheelchair navigating through populated areas. We noticed an improvement in the number and accuracy of the detected bounding boxes compared with the pre-trained COCO models provided by Darknet.

ROS Node

The Python wrappers for Darknet allow us to access the detection and image conversion functionality of the framework. To fully integrate the framework as a ROS node, we must follow the standard ROS procedures for creating a new ROS package. As such, we have created a package[6] that runs Darknet and YOLO that can be downloaded and run seamlessly with ROS.

ROS Topics The ROS node subscribes to the `/image_transport/compressed` and expects to receive images in the compressed JPG format. This is because the Hololens captures video frames and encodes it to reduce the usage on the network bandwidth. These images are converted to numpy arrays which are then converted to Darknet `IMAGE` types for detection.

Darknet produces bounding box co-ordinates and class probabilities for each detection in an image. For every received frame, the node publishes the original image and a list of associated bounding boxes. The bounding box message contains the following information:

[6] https://github.com/alaksana96/fyp_yolo

```
string Class
float64 probability
int64 xmin % Top Left Corner
int64 ymin
int64 xmax % Bottom Right Corner
int64 ymax
```

Listing 5.3: BoundingBox.msg

5.1.3 YACHT Package

The major contribution of this project is in the form of the Yet Another Crowd Human Tracker package for ROS. This package utilizes the Deep SORT algorithm and OpenPose body pose estimation framework to determine the direction of individuals.

ROS Communication As seen in Section 5.1.2, the YOLO object detector node publishes a topic which contains the image and associated bounding boxes. Figure 5.1 shows that both YACHT nodes subscribe to the same output topic. The reason we decided to include the original image in the message and not just the bounding boxes is so that we can be sure the bounding boxes were detected for that frame, without having to subscribe to two separate topics and comparing timestamps on individual messages.

5.1.4 YACHT: Tracker

As explained in Section 5.1.3, the YACHT tracker depends on the Deep SORT algorithm [15]. Using the detections produced by the YOLO node, we assign IDs and match them across video frames to produce a track. These tracking IDs are then sent to the Hololens for visualization.

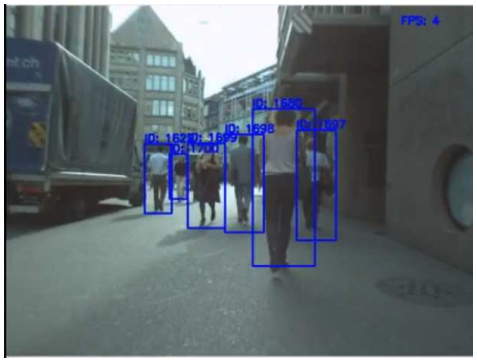
Deep SORT

Modifications We have modified[7] Nicolai Wojke's original implementation of Deep SORT to run on Tensorflow-GPU. In the original, as the number of detections increased, so does the delay in tracking. This issue is more prevalent when tested on the MOT dataset, which has a large number of objects in each frame. Figure 5.7 visualizes the delay on the MOT16-06 video. We can see that the algorithm is operating at a very low FPS, causing it to be delayed in time.

Deep Association Metric Through code profiling, we noticed that the program was spending a lot of time in the generation of feature vectors. Upon inspection, we noticed that this process was run on a CPU bound version of Tensorflow. The `ImageEncoder` class uses a pre-trained deep network that generates the feature vectors for each bounding box. By using Tensorflow-gpu, we were able to run the network on the system GPU, removing the delay.

[7] https://github.com/alaksana96/deep_sort

(a) Output of YOLO. The image frame and bounding boxes are fed to Deep SORT.



(b) Deep SORT is several frames behind since it runs at 4 FPS.

Figure 5.7: Visualization of delay on CPU bound Deep SORT.

```
1  class ImageEncoder(object):
2
3      def __init__(self, checkpoint_filename, input_name="images",
4                   output_name="features"):
5
6          # Tensorflow-GPU
7          gpu_options =
       tf.GPUOptions(per_process_gpu_memory_fraction = 0.2)
8          self.session =
       tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
```

Listing 5.4: Deep SORT Tensorflow GPU modifications.

Trackers & Tracking For each detection, we generate a feature vector using the image pixels within the bounding box. A matching cascade with Nearest Neighbour metric is used to best match the detection with existing confirmed tracks. Some detections will not get matched since the distance to confirmed tracks is above the `matching_threshold`. The algorithm then attempts to match the detections to unconfirmed tracks using a simple Intersection-over-Union metric. These are newly created tracks that have existed for less than the last n frames. If the detection is still unmatched, the algorithm creates a new tracker for the detection and adds it to the pool of unconfirmed tracks.

Any pre-existing tracks which are not matched are checked. If the number of frames since the previous match is greater than the `max_age` parameter, the track is considered dead and is deleted. This is done to prevent the number of tracks from growing infinitely. A Kalman filter is used to update the bounding box states of each track, as well as the time since the last update.

Linear Extrapolation

We experimented with linear extrapolation across frames as a way of inferring the direction travelled by the detection. As the project progressed, we encountered issues

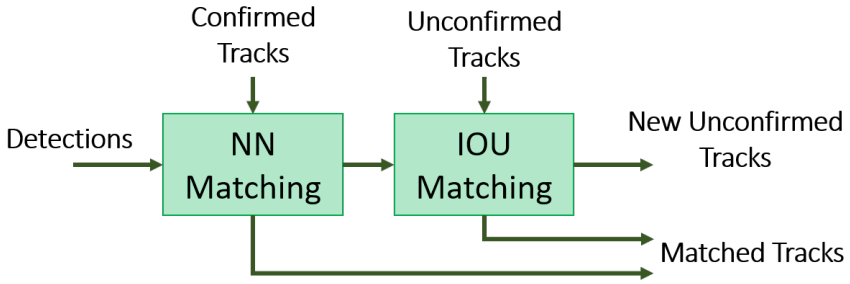


Figure 5.8: Visualization of Deep SORT matching. IOU matching is used as a final check for tracks that are unmatched by feature vector Nearest Neighbour matching.

with this method, as explained in Section 4.2.2. By searching for alternative methods, it was found that the depth camera on the Hololens can determine the distance between the PWU and an object relatively accurately. As such, we abandoned the pure computer vision approach in favour of using the Hololens.

ROS Topic

The decision to use the Hololens depth cameras as a way of determining distance prompted the need for ROS messages to be sent to the device. As seen in Figure 5.2, the tracker node publishes the bounding box and tracker ID to the Hololens. The `BoundingBox` data structure is defined in Listing 5.3, and is the same bounding boxes generated by the YOLO detector.

```
1 BoundingBox boundingBox
2 int64       id
```

Listing 5.5: ROS message structure for BoundingBoxID.msg

5.1.5 YACHT: Direction

The second node in the YACHT package is the direction node, which uses the OpenPose framework to determine if a person is facing the camera or not. Earlier in Section 4.2.2, we outlined the problem of not being able to determine the distance to an object with a regular pinhole camera model. We initially wanted to be able to determine the distance using only a video stream and computer vision techniques. However, we quickly realized that this was beyond the scope of the project, and decided to use the depth cameras on the Hololens.

This section outlines the installation and setup of the OpenPose network [37]. We also explain how we use the key-point detections to determine whether an individual is facing the PWU or not. Furthermore, we also explain how the bottom-up approach of OpenPose differs from the top-down approach that may have been more suitable and the reasons for our implementation choices.

OpenPose

OpenPose is developed and maintained by the Carnegie Mellon University Perceptual Computing Lab. The implementation is made available on Github[8] to encourage body pose estimation research.

Installation & Setup The OpenPose library runs on a modified version of the convolutional neural network framework Caffe [42]. The library is well documented and provides its own instructions on how to set up the library. We direct the reader to the OpenPose GitHub repository if they wish to install the library themselves.

Model As stated in Section 4.2.2, we use the `BODY_25` keypoint estimation model for this project. The documentation states that this model is the fastest when it comes to real-time application, compared to the `MPI_4` or `COCO` models. We also reduce the network resolution to 176×176 to reduce the GPU usage and speed up the keypoint estimation. However, this reduces the accuracy of the detections, as discussed in this section.

KeyPoint Estimation

Due to the bottom-up approach used by OpenPose, the network predicts key-points for individual body parts across the whole image. Through our testing on the MOT dataset, we noticed several points:

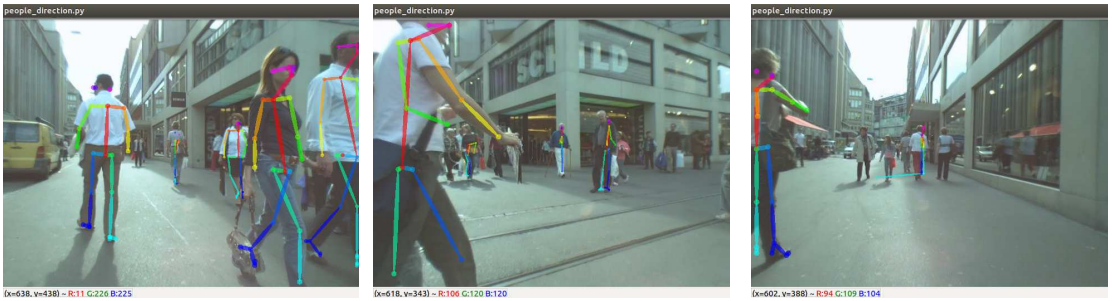
- The model is good at detecting key-points of people close to the camera.
- People who are smaller and further away are not always detected.
- When people are close together or overlap, the keypoint estimation has difficulty differentiating between people.
- The more people in the image, the more network slows down and begins to lag behind the source video.

We highlight the issues in Figure 5.9. The reason for the decrease in accuracy for people further away is due to the lower network resolution. Using a higher resolution allows us to detect people further away, but the delay between the arrival of the frame and the detection is more than 0.5 seconds. As such, to achieve real-time operation, we chose to use a lower network resolution.

Defining Direction

By comparing the relative positions of certain key-points, we can determine if a person is facing the camera or if they are walking away. This information is important since it will allow for better visualization of where a person is walking since people tend to walk in the direction they are facing. This also partially solves the direction problem brought up in Section 4.2.2.

[8] https://github.com/CMU-Perceptual-Computing-Lab/openpose

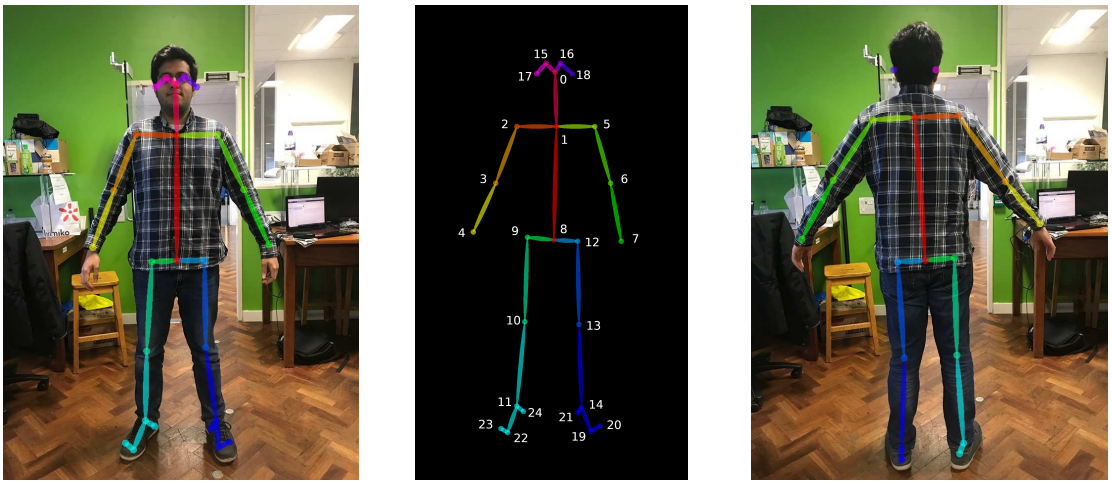


(a) Key-point estimation is most accurate on people close-up.

(b) On people at a distance, the estimation accuracy is limited by the network resolution.

(c) A low network resolution results in difficulty discerning between people.

Figure 5.9: Comparison of keypoint estimation at different scales



(a) Frontal View

(b) Key-point References

(c) Back View

Figure 5.10: Body keypoint estimation of different views. These images show that the network can differentiate between left and right limbs.

Method Pixels are measured from the top left corner of the image, with the x-axis extending horizontally to the right and the y-axis extending downwards. Figure 5.10.b shows the keypoint references for the body. The model can differentiate between the left and right limbs on a person. Figure 5.10.(a,c) show a full frontal and back keypoint detection on a person in the ideal detection position.

Body Part	Key Points
Right Arm	2, 3, 4
Left Arm	5, 6, 7
Head/Spine	0, 1, 8

Table 5.1: Significant key-points for determining whether a person is facing the camera.

We can use the ability to differentiate between the left and right arms to determine if a person is facing the camera. Table 5.1 presents the key-points for the relevant body parts. If the image co-ordinates of the left shoulder is further along the x-axis than the right shoulder, we can predict the direction as facing towards the camera. From testing,

we know this simple method works most of the time. However, problems arise when a person is standing perpendicular to the camera. OpenPose has trouble detecting the torso and predicting the positions of the limbs, and it becomes difficult to decide if they are facing left or right.

Implementation & Detection Matching

As mentioned in Section 5.1.5, OpenPose uses a bottom-up approach by detecting individual body parts across the whole image. We need to match the OpenPose keypoint predictions with existing bounding boxes from YOLO since the tracker node assigns these detections a tracking ID. This is done in Listing 5.6

```
def matchDetectionAndPose(self, detections, poses):
    for pose in poses:
        # Check torso, right/left shoulder
        torso, rshoulder, lshoulder = pose[1], pose[2], pose[5]

        for bbox in detections:
            if( self.withinBB(bbox, torso[0], torso[1]) or
                self.withinBB(bbox, rshoulder[0], rshoulder[1]) or
                self.withinBB(bbox, lshoulder[0], lshoulder[1])):

                if(rshoulder[0] > lshoulder[0]):
                    directionTowardsCamera = False
                else:
                    directionTowardsCamera = True

                publishDetectionDirection()
                break # Once matched, move onto next pose
```

Listing 5.6: Direction and Detection Matching code in people_direction.py

ROS Topic

The direction node publishes the bounding box and direction to the Hololens. The `BoundingBox` data structure is defined in Listing 5.3, and is the same bounding boxes generated by the YOLO detector.

```
BoundingBox boundingBox
bool        directionTowardsCamera
```

Listing 5.7: ROS message structure for BoundingBoxDirection.msg

5.2 Hololens Unity Application

The Hololens is the central device in the overall system, acting as an intermediary between the HDD system and ARTA. We rely on the AR capabilities of the device to render visualization holograms to indicate to the user potential collisions. The depth camera and other sensors are used to perceive the distance of detected objects which are communicated to ARTA for reactive assistance. Most importantly, the front facing camera is the main visual input for the whole system, beginning the whole image processing pipeline. Without a live video stream from the camera, object detection and direction inference would be impossible. As such, the initial phase of the project was dedicated to producing a reliable video stream to an external computer. Figure 5.11 shows the two separate parts of the Hololens application, the camera stream, and the holographic world.

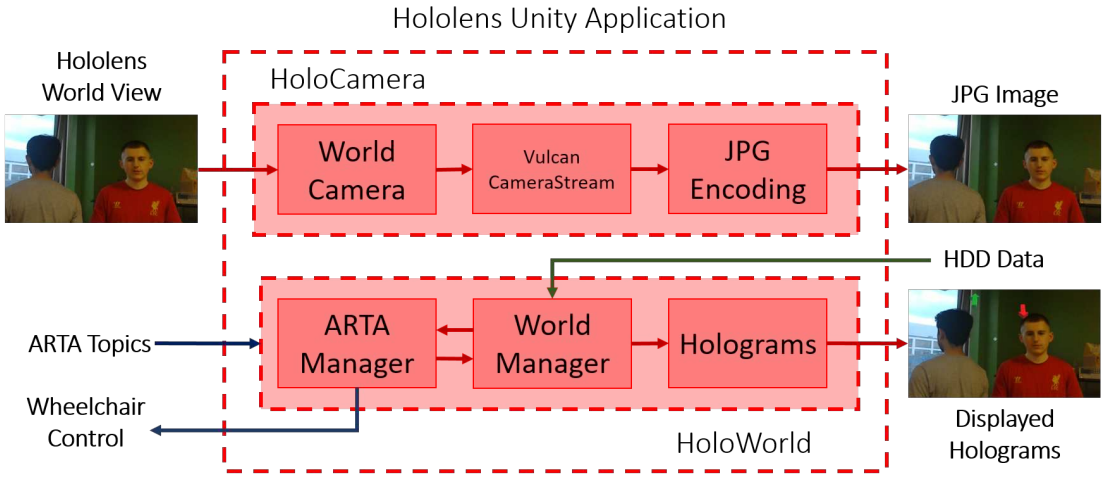


Figure 5.11: Components of the Unity application running on the Hololens. We divide the program into two sub-programs, HoloCamera and HoloWorld.

Due to the importance, we begin this section by describing the implementation of the video camera stream. We explain the libraries used, the workarounds required to produce the stream, and the limitations of using a Unity-based approach. Afterwards, we describe the Unity application responsible for creating holograms, controlling ARTA, and how detected objects are placed in the Unity world. Finally, we explain the interaction between the Hololens and ARTA.

5.2.1 Hardware & Software Dependencies

Hardware

Microsoft Hololens The Microsoft Hololens is the world's first fully untethered holographic computer. The device can render 3D holograms in the world surrounding the user by displaying them on a set of see-through holographic lenses. The device is equipped with a multitude of sensors which we outlined in Section 2.5.2. The device

also supports gesture recognition for controlling the device, as well as voice input. The device is also able to connect to Wi-Fi networks, allowing it to stream data to and from other devices. We provide a complete device specification in the Appendix.

(a) The Microsoft Hololens with the see through holographic lenses.

(b) The device is worn on the head, resting on the bridge of the nose, similar to glasses.

Figure 5.12: The Hololens is larger compared to its competition. However, the increased number of sensors and wider spread usage makes it an ideal AR device for this project.

Software

Universal Windows Platform The Universal Windows Platform (UWP) allows for applications that can be developed to run on any device running Windows 10. It gives the developer access to a set of standard APIs that make it easier to access data from any device. Software Development Kits (SDKs) are used to extend the capabilities of an app for specialized devices such as the Hololens.

Unity Unity is a cross-platform game engine used to develop 3D, 2D VR & AR visualizations for applications such as games, engineering or assistive robotics, among others. The primary programming language for the engine is C#, and the engine has built-in Windows Mixed Reality and Hololens support, allowing for easier development of applications for the Hololens.

Development Tools For this project, we used the following setup for development:

- Microsoft Hololens running Windows 10
- Laptop running Windows 10 Developers Fall Edition
- Unity Editor 2018.1.6f1 (64-bit)
- Visual Studio 2017 Community Edition

The application produced in the Unity Editor and Visual Studio 2017 is compiled and deployed to the Hololens as a UWP Unity application that can be accessed and run from the device's heads up display menu.

5.2.2 Hololens Locatable Camera

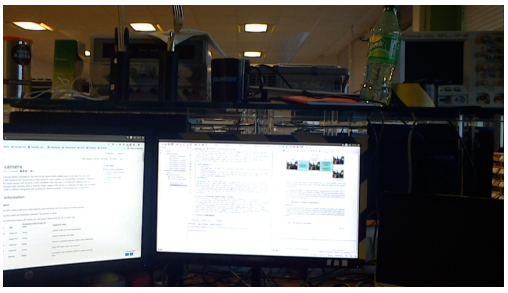
The images captured by the front facing camera is meant to be a representation of what the user is seeing. In reality, the field of view (FOV) of the front facing camera is greater than the actual FOV of the device, meaning the camera can see more of the world than the user can.

Specifications

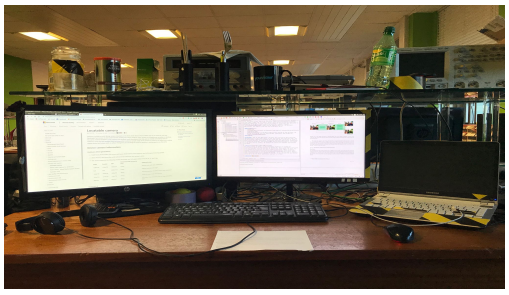
The front facing camera is a two megapixel photo/ HD video camera with auto white balance, auto exposure, and a full image processing pipeline. Due to the real-time requirement of the project, we chose to use the smallest video resolution possible in order to achieve the fastest frame rates. This limits the resolution of the captured video frames to 896×504.

Limitations

The greatest limitation of the Hololens is the field of view. The FOV of the front facing camera is 48 degrees wide, allowing it to capture images of the surrounding world. However, compared to a standard mobile phone camera, the front facing camera provides a cut-off view of the world. We highlight this issue in Figure 5.13, which shows the reduced FOV of the device. Another issue is that despite the image processing the pipeline, the Hololens camera produces overexposed images, especially when the front facing camera is pointed towards illuminating objects or when light is only coming from one direction.



(a) Image captured by Microsoft Hololens. Notice the overexposure in the image.



(b) Image captured by standard IPhone Camera under the same light conditions.

Figure 5.13: These images were taken from the same position. The Hololens has a reduced FOV and is unable to capture the whole desk.

Furthermore, due to the positioning of the holographic lenses, the users FOV is estimated to be approximately 29-30 degrees wide and 17 degrees high. This is even less than the front-facing camera and limits what the user can see in the surrounding area.

5.2.3 Hololens Video Camera Stream

For this project to begin, it was essential to stream the front-facing camera on the Hololens to another PC. Once it had been proven that this was possible, work on the

rest of the project could begin. As such, the first month of the project was spent comparing different camera streaming methods. As mentioned in Section 4.3.2, we briefly spent some time attempting to use the HoloLensForCV library from Microsoft. However, after speaking with several members of the PRL, it was decided the best method would be to use the Unity application approach as a base.

Image Capture

The Unity engine on the Hololens has APIs for the front-facing camera, giving developers access to information such as the resolution of the captured images, the camera intrinsics, and the projection transform. However, Unity only allows developers to take a photo or video recording that is then saved onto the filesystem. The media can then be loaded from disk into Unity for streaming. This process is slow and time-consuming, due to the large overheads associated with writing media to and from disk. From experimentation, we deemed this method too slow for our use case and sought out an alternative.

Vulcan Technologies HoloLensCameraStream is a Unity plugin[9] developed by Vulcan Technologies that makes the Hololens front facing camera frames available inside the Unity app in real time. The original library was designed to give developers the ability to show a preview of what the Hololens camera sees within the application. The ability to access the raw frames inside saves the application from having to write the frame to disk and loading it again. Unfortunately, the library has not been maintained for newer versions of Unity with the last stable version being Unity 2017.3.1f . Furthermore, these older versions of Unity are incompatible with the ROS# library we use to communicate with other ROS nodes. As such, we updated the plugin for Unity 2018.1.6f1 by modifying the offending code and updating obsolete functions to the newer 2018 Unity API.

Image Capture Pipeline We briefly explain the image capture pipeline in the Unity application outlined in Figure 5.14. An instance of the updated Vulcan Technologies *Camera Stream Helper* is added to the scene which provides access to the front facing camera video stream. We set the camera parameters to maximize the frame-rate by lowering the resolution.

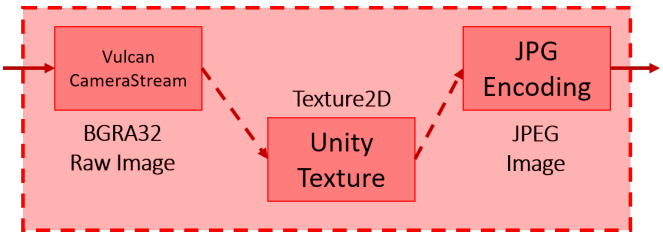


Figure 5.14: The image capture pipeline involves an intermediate Texture2D Unity state.

[9] https://github.com/VulcanTechnologies/HoloLensCameraStream

The captured frames are in the Microsoft standard `BGRA32` pixel format, which has 32 bits per pixel. The four channels (blue, green, red, and alpha) are each allocated 8 bits per pixel. The raw pixel format creates large filesizes unsuitable for streaming over the network. As such, it becomes necessary to compress the images into a PNG or JPEG format. Unity can convert *textures*, a data structure for rendering images inside applications into a compressed image format to be saved to disk. We leverage this ability to compress the raw BGRA32 bytes into a JPEG image ready for network transfer.

Image Compression

Unity has built-in image compression techniques for saving textures to disk. JPEG images are generated through a lossy compression technique. The degree of compression can be adjusted, allowing a trade-off between image quality and storage size. Since these images are to be streamed over the network, we want to limit the size of the files not to use too much bandwidth. On the other hand, we want the quality of the image to be high enough so that we can conduct image processing in the HDD system. Through experimentation, we chose to set the image quality to 50%, the middle ground between quality and filesize.

Unity Threading Applications running on the Unity engine support threading, but due to many Unity types not being threadsafe, certain actions must be done in the main thread of the application. The main thread is responsible for all object interactions in the scene, as well as the rendering of holograms and other visualizations. On the other hand, tasks such as image capture by the Vulcan Camera Stream can be done in separate threads which are switched to from the main thread. We visualize the threading process in Figure 5.15.

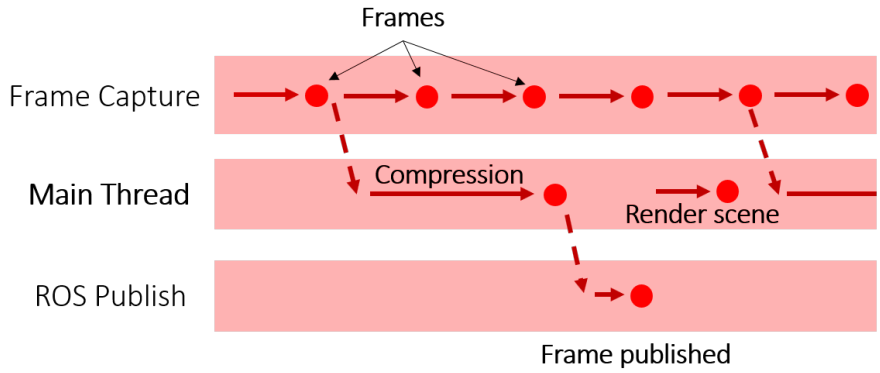


Figure 5.15: Image compression is a time consuming operation that must be done in the main thread. Only after compression is done can the holograms be rendered, limiting the application to 5 FPS.

Limited Frame-rate An issue comes up with the way Unity implements image compression. The JPEG encoding reads the raw image bytes from a `Texture2D` object in the Unity scene. This action must be conducted on the main thread since updating

textures involves object interactions. From code profiling, we determined that it takes approximately 0.2 seconds to compress a captured frame. As such, the rendering of the scene can only be done after image compression. This results in a slight latency of the visualizations viewed by the PWU.

5.2.4 ROS Communication

As explained in Section 4.3.1, we use the ROS# library originally released by Siemens for our Unity to ROS communications. However, the original ROS# library was not developed for Unity applications running on UWP. As such, we used a variant of the library[10] developed by David Whitney, a Ph.D. student at Brown University.

Modifications

The UWP version of ROS# has been modified to only use APIs available on UWP devices such as the Hololens. Upon adding the ROS# plugin to our Unity application, we encountered compatibility issues since the plugin was developed for Unity 2018.2 and above. We were forced to use 2018.1.6f1 since it is the most recent version of Unity, where the HoloCameraStream library works, and it is possible to access the camera frames inside the Unity application. As such, we had to backport the ROS# library by changing function calls to newer API features not available in our version of Unity.

ROS Topics

Figure 5.16: ROS topics in and out of the Unity application. Each topic carries data used by different parts of the overall system.

Compressed Images The raw video frames captured by the front-facing camera are compressed into JPEG frames that get published in the form of `CompressedImage` messages across ROS topics. Within the Unity application, we have setup `CompressedImage` publishers which are called when a compressed video frame is ready to be read into a message and published.

[10] https://github.com/dwhit/ros-sharp

HDD Topics In Section 5.1.4 and Section 5.1.5, we have explained the ROS messages published by the HDD system. The Hololens Unity application subscribes to these topics and receives the messages to transform the image co-ordinates into real world positions.

5.2.5 Hololens World

Mixed reality applications can place holograms in the world on real objects. This involves precisely positioning the holograms in places in the world that are meaningful to the user, which in the case of this project, is on detected people walking in the surroundings. Upon receiving messages from ARTA and the HDD system, the Hololens must parse the data to be able to update its internal understanding of the surrounding world. This section explains the other half of the Unity application, which is responsible for communication between devices, positioning of objects in the surrounding world, and visualizing detections using holograms.

Hololens World Manager

Every Unity application has *scenes* where developers can place *GameObjects* which render holograms, recognize gestures, and communicate with other devices. A scene can be thought of as the Hololens' understanding of the world around it. The World Manager acts as the main program in the scene, receiving messages from the HDD system and ARTA, keeping track of objects in the surroundings and the position of the Hololens and ARTA in the real world.

Communication The World Manager maintains a collection of all the publishers and subscribers on the Hololens. Figure 5.16 outlines the flow of information in and out of the Unity application. By maintaining a central communication hub, specialized scripts such as the Hologram Manager or ARTA Manager can communicate with the other devices without having to create publishers or subscribers.

Spatial Mapping Through spatial mapping, surfaces can be detected and objects rendered on top of tables or other solid objects. The Hololens has built-in cameras that continuously scan the surrounding environment, building up a virtual world geometry of the real world objects. The World Manager provides a central spatial mapping class that all scripts can access so they have the same understanding of the surrounding world.

HDD Data to GameObjects

From Figure 5.16, we can see that the Unity application subscribes to two topics from the HDD system:

- `/yacht/output/detectionids/`
- `/yacht/output/detectiondirections/`

These are the outputs of the two YACHT nodes described in Section 5.1.3, where we also describe the format of the output messages. The frames captured by the Hololens are sent to the HDD to be processed. From the frames, the HDD replies to the Hololens by sending bounding box image coordinates of the detections, as well as a tracking ID and whether the detected person is facing the camera.

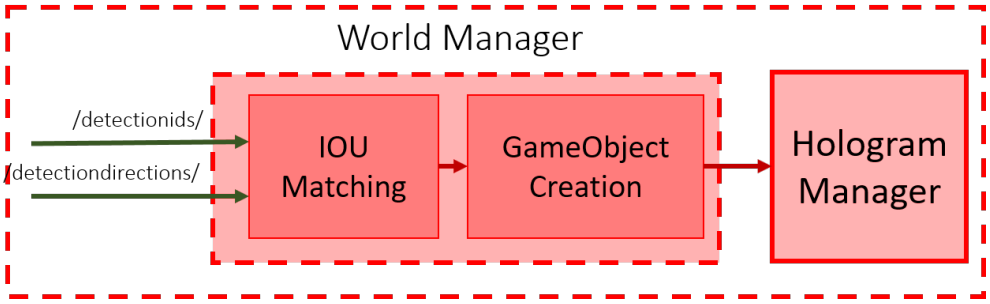


Figure 5.17: The Intersection-over-Union metric is used to compare bounding boxes for matching. The higher the metric, the more overlap between the boxes.

Matching Bounding Boxes Due to the design of the YACHT package, two different nodes determine the tracking ID and direction of a detection. This allows the processing to be done in real-time by running on separate threads. Since both YACHT nodes subscribe to the bounding box detections from the YOLO node, we can match IDs and directions by comparing the bounding boxes contained in each message.

```
1  private float IntersectionOverUnion(BoundingBox a, BoundingBox b)
2      {
3          int xmin = Math.Max(a.xmin, b.xmin);
4          int ymin = Math.Max(a.ymin, b.ymin);
5          int xmax = Math.Min(a.xmax, b.xmax);
6          int ymax = Math.Min(a.ymax, b.ymax);
7
8          int interArea = Math.Max(0, xmax - xmin + 1) *
       Math.Max(0, ymax - ymin + 1);
9
10         int aArea = (a.xmax - a.xmin + 1) * (a.ymax - a.ymin + 1);
11         int bArea = (b.xmax - b.xmin + 1) * (b.ymax - b.ymin + 1);
12
13         float iou = interArea / (float)(aArea + bArea -
       interArea);
14
15         return iou;
16     }
```

Listing 5.8: IOU metric C# implementation.

We use the Intersection-over-Union (IOU) metric to match the bounding boxes by comparing the amount of overlap. This was done since there may potentially be network delays between the received messages. If latency is involved, the bounding boxes

from the detection may not match up precisely, since one message may be compared with another message from the frame before. By using IOU, we can compare the metric to a threshold to determine a match.

Creating GameObjects We use GameObjects to represent the detected people in the surroundings. Before we can place the GameObjects of the people in the scene, we first need to determine the real world positions from the position in the image frame. This is done by un-projecting the image and using a coordinate frame transforms to convert the point in the Camera coordinate system to the World coordinate system. We highlight the process in Figure 5.18.

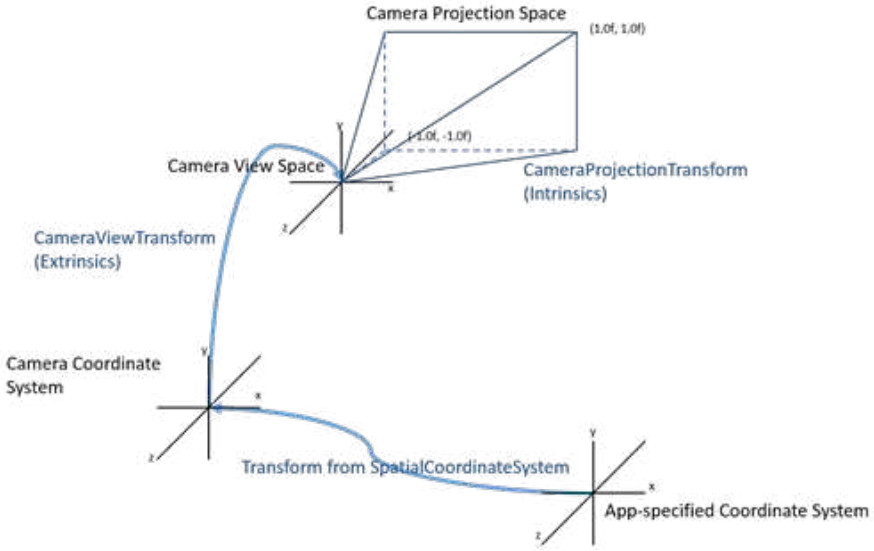


Figure 5.18: The front facing camera projects the surroundings onto a 2D plane. We can determine the real world position of a pixel by reversing the projection and using frame transforms [29].

Fortunately, the Vulcan Technologies library used to capture the frames has a function that converts pixel locations to world space coordinates. Using the `camera2World` matrix transform and camera projection matrix, the `PixelCoordToWorldCoord()` function determines the real world position of a pixel. We use the centre of the bounding box centres to represent the position of the detected people, and we place the GameObjects at the initial world space position determined by the function.

Hologram Manager

The Hologram Manager is the manager of visualizations rendered on the holographic screens. The GameObjects representing detected people created by the World Manager are used to determine the initial positions of the holograms, but the Hologram Manager uses ray casting and spatial mapping to correct the positioning of the GameObjects. The holograms are then rendered by the manager in their correct positions to communicate to the PWU if people are walking towards them or not.

Holograms We use *prefabs*, 3D holographic models we created in the Unity editor to create the visualizations. When the GameObjects representing people are created, they are invisible to the PWU, since we are not rendering any textures on the objects. By attaching a prefab to the GameObject, we essentially place a hologram at the position of the GameObject in 3D world space. We have created two prefabs, a green and red arrow. The green arrow indicates to the PWU that the person is moving away or in the same direction as the PWU is facing, while a red arrow represents a person walking towards the PWU. We show these prefabs in Figure 5.19, as well as the blue cursor that indicates to the PWU where they are looking.

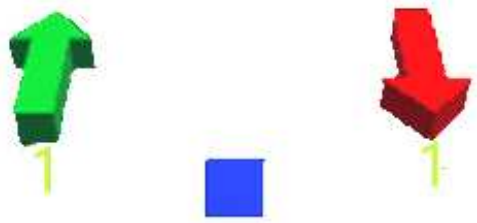
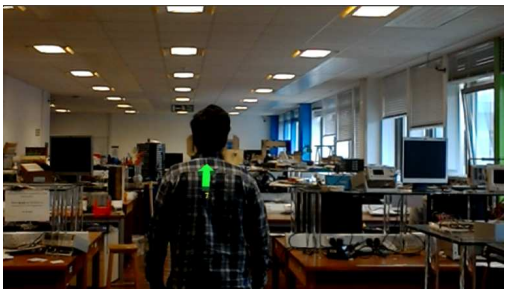
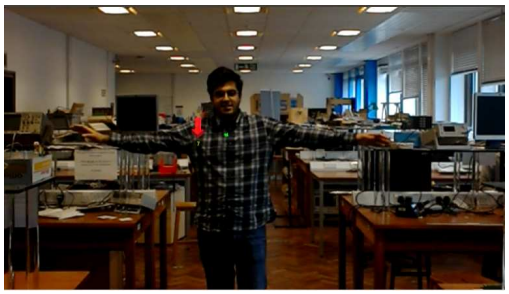


Figure 5.19: The Green and Red Arrow prefabs. The number under the arrow is the associated tracker ID of the object. This allows us to track objects in the surroundings.

To improve the positional accuracy of the holograms, we utilize the spatial mapping capabilities of the Hololens. By using the `Raycasting` function in Unity, we can project a ray from the Hololens in the direction of the GameObject. When the ray hits a surface detected by the spatial mapping class, we render the hologram on the surface and move the GameObject to that position. This tries to ensure that the holograms are rendered on the detected person, as seen in Figure 6.5.



(a) Green Arrow hologram rendered on a person walking away.



(b) Red Arrow hologram rendered on a person walking towards the camera.

Figure 5.20: Images captured from the Hololens showing what the PWU would see when wearing the device.

ARTA Manager

We create a GameObject to represent the powered wheelchair within the Unity application. The ARTA manager subscribes to topics published by the ROS nodes on the device, shown in Figure 5.16. Using the messages received from the powered wheelchair,

it is possible for the application to understand the PWU's control intentions and to make decisions reactively. We explain the collision avoidance reactive control system in the next section of this report.

5.3 ARTA Powered Wheelchair

The Assistive Robotic Transport for Adults is a powered wheelchair developed by the Personal Robotics Lab at Imperial College London. A PWU can control the device through a joystick that converts the user issued commands into motor signals. We communicate the wheelchairs velocity and PWU control signals to the Unity application running on the Hololens and using the messages from the HDD system; we developed a reactive control system to detected people in the surroundings of the PWU.

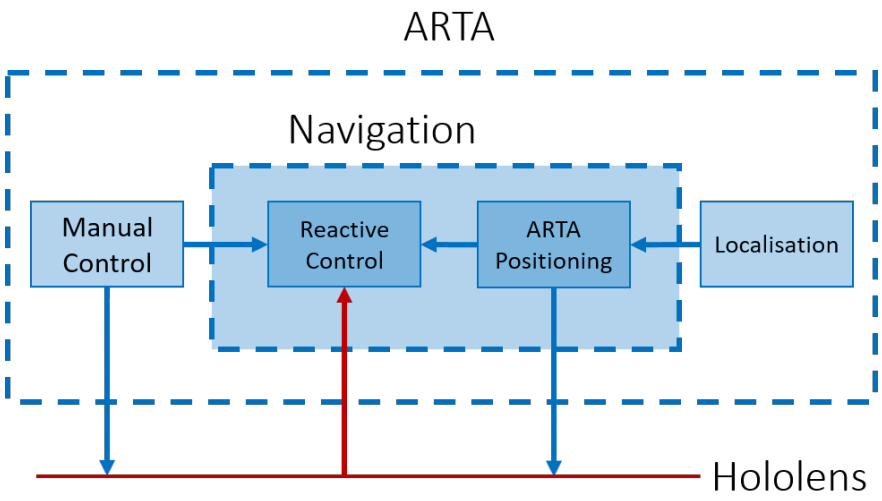


Figure 5.21: The ROS nodes running on ARTA. We utilize PRL ROS packages as a base and communicate with the Hololens for reactive control.

5.3.1 Hardware & Software Dependencies

Building a powered wheelchair with the required sensors for localisation and mapping is a time-consuming process, well beyond the scope of this project. We rely on the sensors already attached to the wheelchair, as well as the ROS packages developed internally for the conversion of joystick commands to control signals and communication between ROS nodes.

Hardware

The following sensors are fixed to the powered wheelchair for navigation and localization purposes. The laptop is used as the main control system running the ROS nodes that utilize the sensor data.

- Arduino Uno (Joystick to motor signal conversion).
- 2 Hokuyo URG-04LX-UG1 laser scanners.
- SICK LMS200 rangefinder.
- Phidgets spatial 3/3/3 IMU.
- Laptop running ROS nodes.

Software

The PRL has several GitHub repositories dedicated to the ROS nodes running on ARTA. These ROS packages cover the robotic aspect of the powered wheelchair, from using the sensor data for localization and mapping of the surroundings, to receiving control signals from the joystick or other exotic control interfaces. We utilize these repositories to build the base ARTA system that is needed to control the robot using the joystick.

5.3.2 PRL ROS Packages

A large robotic device, such as ARTA requires a lot of software to control. As such, the PRL has divided the ARTA functionality into several ROS packages, each responsible for a certain aspect of controlling the powered wheelchair. We utilize these packages as a base and contribute our additional reactive control module which works in conjunction with detections from the Hololens.

ARTA

The ARTA ROS package is the main parent node that encompasses the other packages we describe in this report. The package is responsible for the basic manual control and sensor node activation. It provides an interface via ROS messages for other nodes to communicate with, such as the localisation or navigation nodes. Furthermore, the package is responsible for the Gazebo simulation of the device, such as the model of the wheelchair and control systems.

Localisation

The PRL Localisation package is an internally developed library for the robots in the PRL lab. It contains various ROS launchers for common SLAM and localisation packages such as the `hector_mapping` SLAM approach that utilizes laser scan data for mobile robots. The package also contains maps for use with the `amcl` package for localisation of the robots. The members of the PRL have constructed maps for commonly used testing areas, such as the hallway outside the PRL lab on the 10th floor of the EEE building. We leverage these pre-existing maps for testing and evaluation later in this report.

Navigation

In addition to the localisation package, the PRL has developed a ROS package for assistive navigation of the mobile robots in the lab. The package encompasses obstacle avoidance, autonomous navigation as well as shared control. Using the sensor data and localisation techniques from the other packages, we developed our reactive control system with this as a base.

5.3.3 Hololens Communication

As shown in Figure 5.16, the Hololens Unity application requires data from ARTA to build a Unity GameObject of the powered wheelchair. We utilize ROS topics and messages to communicate to the Hololens the position of the wheelchair in the world, the PWU's joystick input controls, and the current wheelchair velocity. Additionally, Figure 5.21 shows the communication from the individual ARTA nodes and highlights the control messages received from the Hololens.

5.3.4 Reactive Control

The goal of this project was to build an augmented reality system that would assist PWU in their day to day navigation tasks. Detection of people in the surroundings and visualization of their direction is useful to the PWU, but reactive control for assistive navigation and object avoidance can be seen as a direct use of the people detection. We contribute a system that uses the people detection data from the HDD system and front-facing camera of the Hololens to manipulate the PWU manual control signals for wheelchair control.

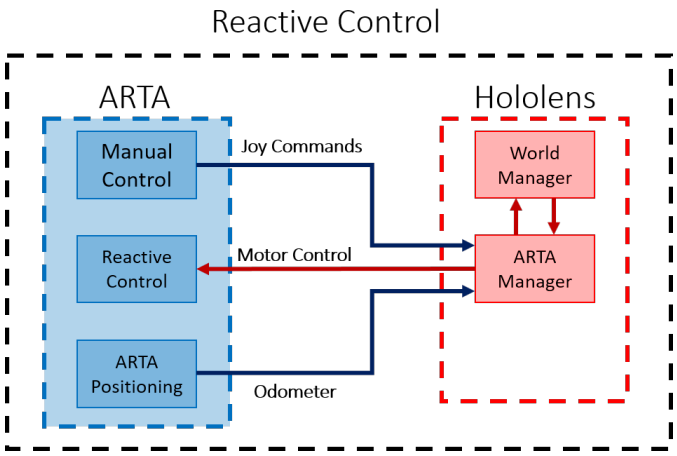


Figure 5.22: The Reactive Control system spans both ARTA and the Hololens, due to its dependency on ARTA sensor data as well as the Unity converted HDD system GameObjects.

We have left the explanation of the reactive control system for last since it spans both ARTA, the Hololens Unity application. We go over the flow of information through the systems, how they contribute to the final reactive control decisions made by the ARTA

manager on the Hololens and how it is translated to motor signals and movement of the powered wheelchair.

ARTA

Manual Control The powered wheelchair is controlled using a joystick, which the PRL have attached a circuit board and Arduino UNO that converts the user inputs into control signals. The signals are published on the `/arta/arta_joystick` topic, which are sent to the nodes responsible for turning the wheels on the device. Furthermore, the localisation and navigation nodes also subscribe to these topics, since libraries such as the obstacle avoidance nodes require the user input signals. As shown in Figure 5.22, the Hololens also subscribes to the topic via the ARTA manager.

ARTA Positioning For this project, we have limited the scope to navigating the wheelchair in the hallway outside of the Personal Robotics Lab. We have chosen this since the PRL have already built up height maps of the hallway, allowing the localisation packages to determine the position of the wheelchair in the world accurately. By publishing the odometer readings to the Hololens, we can determine the rotation of PWU relative to their head rotation, solving the issue of not knowing if a detected person is directly in front of the wheelchair brought up in Section 4.3.3.

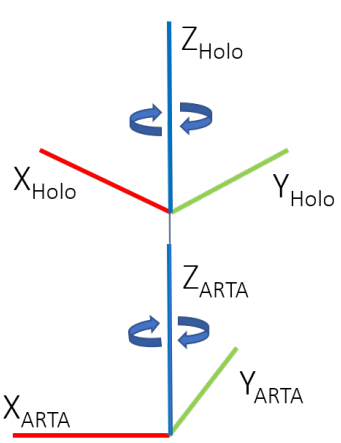
Hololens

World Manager As explained in Section 5.2.5, the World Manager creates GameObjects representing human detections in the Unity scene. By maintaining a map of the positions of these objects, the reactive control system can use spatial mapping to determine the distance between objects in the real world.

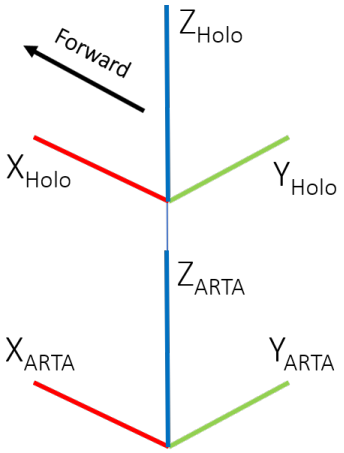
ARTA Manager The ARTA manager forms the central part of the Reactive Control system. It subscribes to the control and odometer readings from the powered wheelchair and uses it to determine if a detected person is in front of the wheelchair and poses a collision risk. Furthermore, upon determining a risk, it manipulates the PWU control inputs to minimize the risk of a collision by lowering the speed of the wheelchair relative to the distance and direction of the person.

ARTA Hololens Alignment

As brought up in Section 4.3.3, the major issue we are faced with is determining if a person detected by the Hololens is in front of the wheelchair, or if the PWU is looking to the side, and the person is not in the wheelchairs forward trajectory. We visualized this problem in Figure 4.10, and show the alignment of the Hololens and ARTA frames in Figure 5.23.a. Although the PWU can tilt their head and rotate the Hololens about the x and y-axis, we have simplified the model and assumed that the PWU will only rotate about the z-axis. Furthermore, through testing, we have determined that the Hololens can only rotate around the z-axis between -90 and $+90$ degrees. The PWU is unable to swivel their head further, due to human biomechanics and the headrest of the wheelchair.



(a) The Hololens frame can rotate between -90 and +90 degrees, while ARTA can rotate 360 degrees.



(b) We calculate the offset between the Hololens and ARTA by aligning the frames during calibration.

Figure 5.23: The alignment of the ARTA and Hololens. We note that both frames can rotate about the z-axis.

However, ARTA is limited to rotations about the z-axis, since the powered wheelchair moves within the 2D plane of the floor. As such, we can determine the initial alignment between ARTA and the Hololens.

Alignment Process Upon starting the Hololens Unity application and ARTA nodes, the Hololens subscribes to the odometry topic published by ARTA. By accessing the pose message contained within the odometry message, we can determine the orientation of the robot in space. The PWU wears the Hololens and sits in the powered wheelchair, facing directly forward. After 5 seconds, the ARTA manager aligns the Hololens to ARTA by calculating an offset between the current ARTA rotation and Hololens rotation measured by the IMU of the device. We show this alignment in Figure 5.23.b. This offset is used to determine whether the PWU is looking forward or to the side.

Unity Distance to Objects

The World Manager uses spatial mapping and raycasting to correct the world positions of the GameObjects representing the detected people. The Hololens has an optimal hologram placement range of between 2 meters and 5 meters. This means after correction; we can be more certain about the distance to the GameObjects if they are within this range. Unity keeps an internal spatial coordinate system of the world, and by keeping track of the Arta/Hololens GameObject position, we can calculate the distance between the device and detected people in the scene.

Reactive Control Algorithm

The ARTA Manager subscribes to the PWU joystick inputs which are translated to ROS `Twist` messages. These messages represent the linear and angular velocities of the

wheels of the powered wheelchair. The forward kinematics of the wheelchair is determined by the linear velocities of the left and right wheel of the wheelchair. To turn the wheelchair, we use angular velocities representing rotations in the z-axis. These messages are received by the Hololens and interpreted to determine the current trajectory of the wheelchair.

At the same time, the ARTA Manager searches through the Unity scene for GameObjects in front of ARTA/Hololens and groups them by the direction component of the GameObject from the HDD system. The reactive control system iterates over the GameObjects with directions coming towards the PWU first, since these are more likely to cause a collision. The system then calculates the distance between the PWU and the detected person.

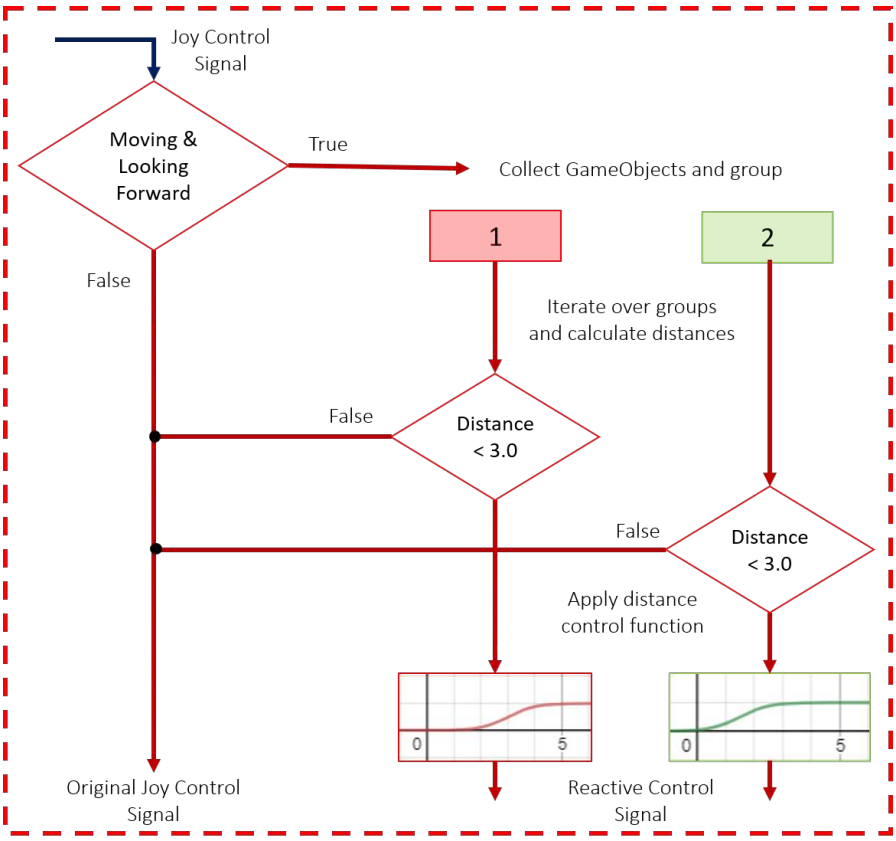


Figure 5.24: A visualization of the Reactive Control system and the corresponding distance speed scaling functions.

As aforementioned, the ideal placement of holograms for distance accuracy is between 2 to 5 meters. Therefore, the reactive control system checks if the detected person is less than 3 meters away. We chose this value to give the whole system time to detect people and make a reactive decision before a collision occurs. We also check if the angle between the person and the powered wheelchair is within 15 degrees of the current trajectory. If the detection falls within both parameters, the system deems the person a collision risk, and the reactive control system takes over.

Upon detecting a collision risk, the reactive control system attempts to avoid a collision by reducing the speed of the powered wheelchair. We have used two scaling functions that reduce the velocity of the wheelchair as the distance to the object decreases. If the person is directly in front of the wheelchair, the function brings the speed of the wheelchair to zero and prevents the device from moving forward until the collision risk has moved out of the way. We show these functions in Figure 5.25, which for people facing the wheelchair $f(x) = \frac{1}{1+e^{-2x+6}}$ and for people walking away $f(x) = \frac{1}{1+e^{-2x+3}}$. We chose to use different scaling since the wheelchair should slow down more quickly if a person is walking towards them.

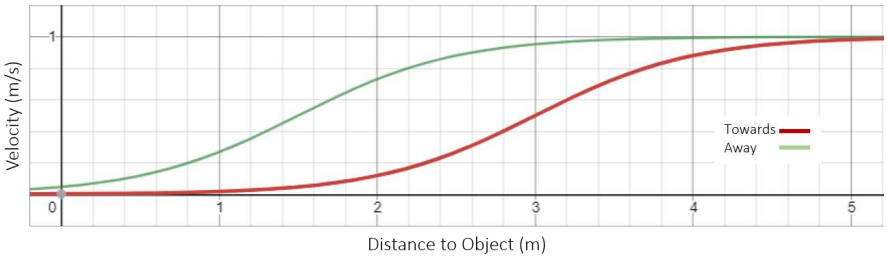


Figure 5.25: The velocity scaling functions relative to the distance to the object. We note that the red function begins decreasing when the detected person is further away. This is to compensate for the person walking towards the PWU.

Finally, these modified control signals are published to ARTA. The main ARTA node receives these control signals and passes it on to the node controlling the wheels of the device.

5.4 System Summary

To summarize, we have implemented a system which utilizes the Microsoft Hololens as a form of visual input by way of the front-facing camera. The images captured by the camera are processed on an external computer. The processing begins with people detection, and the detected objects are fed into an object tracker and body pose estimation system to determine their directions. The bounding box detections and direction estimations are sent to the Hololens for spatial mapping, building a Unity scene with GameObjects that represent the detected people. Using the augmented reality capabilities of the Hololens, we render holographic visualizations for the PWU to see, indicating whether a person is walking towards them or not. Since the PWU is controlling the powered wheelchair manually, we use the reactive control system to monitor the joystick inputs from the PWU and determine whether the system should take over control from the user. If the system decides there is a collision risk in front of the powered wheelchair, it slows down the wheelchair based on the distance between the PWU and the detected person.

Chapter 6

Testing & Results

This chapter details the testing that was carried out to assess the performance of different parts of the system, such as the human detection and direction, as well as the powered wheelchair and Hololens system as a whole. We outline the test setups used to evaluate the performance of the systems, as well as the results of the tests and what they imply about the implemented system.

6.1 Human Detection and Distance

This section is concerned with testing the Human Detection & Direction system and the Hologram GameObject placement. To ensure the system will be able to detect real people moving around in the surroundings, it is necessary to test the system detecting people at different distances and the directions they are facing. We also want to test the accuracy of the spatial mapping system in terms of correcting the real world positions of the GameObjects representing the detected people. We utilize the Microsoft Hololens and the HDD system for this test.

6.1.1 Test System Description

As explained in the Implementation chapter of this report, the front-facing camera uses ROS topics to stream video frames to a partner computer the HDD system is implemented on. The HDD processes the frames and detects people and determines whether they are facing the PWU or not. The bounding box of the detections and directions are sent back to the Unity application. Initially, the application converts the pixel coordinates of the detections to corresponding world coordinates. We then use the spatial mapping and ray casting capabilities of the Hololens to correct the world position distances of the holograms representing the detected people.

6.1.2 Test Setup

We set up a testing ground in the ICRS Lab on the 5th floor of the EEE building. We marked out points at 1-meter intervals, which indicate where the target people should stand. We asked the person wearing the Hololens to sit down in a chair to emulate the position and height a PWU would be at when sitting in the wheelchair. Figure

6.1 shows the experimental setup in the lab. For this step, all subjects were stationary, except when the target person moves between the markings.

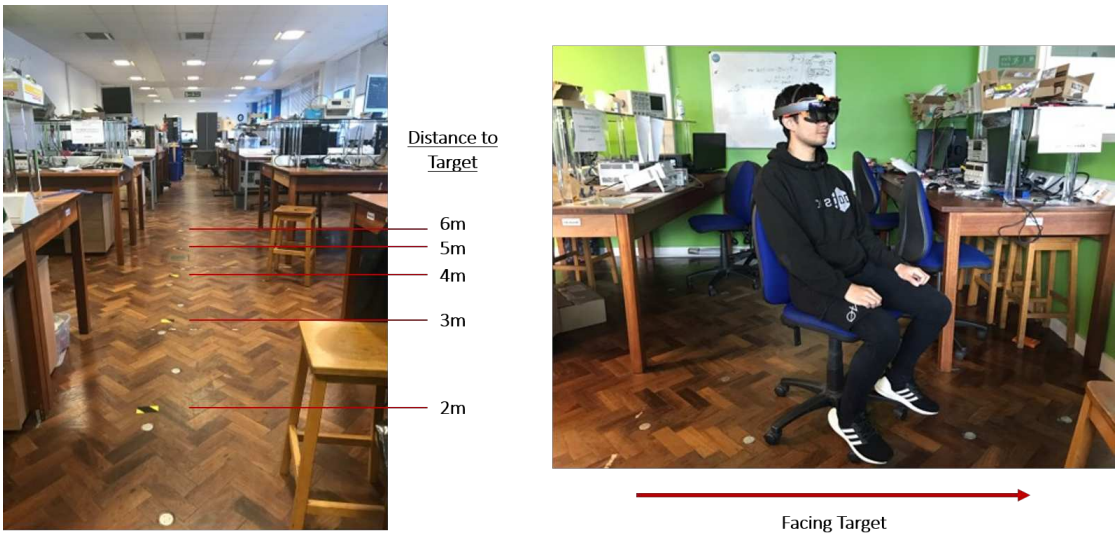


Figure 6.1: The experimental setup used to test the HDD and Hololens spatial mapping accuracy. We asked the people acting as targets to stand at 1 meter intervals while a person sitting down wearing the Hololens looked at them using the front facing camera.

We modified the Hololens Unity application to display holograms showing the distance between the Hololens and the target person in meters. We asked the person wearing the Hololens to call out the value the hologram displayed as the target person stood at different markings. To verify the readings, we recorded the Hololens display using the Windows Device Portal and watched them over after the test. Recording the Hololens view also allowed us to see how the distance reading change as the target person moves.

6.1.3 Test Procedure

We asked three different people to sit in the chair and wear the Hololens. Before commencing the test, we allowed the people to learn how to wear the Hololens properly, ensuring the Hololens was correctly placed on the bridge of the user's nose. We also allowed them to do several test detections to familiarise them with how the distance hologram is rendered. This was done so that each participant would be able to detect the target person and call out the distance displayed by the hologram.

The selected people all had different amounts of experience with the Hololens. One user was a complete novice, putting the Hololens on for the first time to participate in the test. One was an intermediate, having worn the Hololens a few times during the development of this project, and the last person was more experienced with the Hololens and how the HDD system would detect the target people.

We asked the target person to face forward at each marker, before turning their back and facing away from the Hololens user. This was done to check if there were any differences between the placement accuracy of the GameObjects for different orientations of the person.

6.1.4 Results

We show the results of the experiment in Figure 6.2, which compares the accuracy of hologram placement after spatial mapping and ray casting with the true world position of the detected person. Each test subject was given several seconds to adjust their head positions to try to get a detection. Despite this, none of the three test subjects were able to get holographic distances for a target at 6m or further. This may indicate that the system was unable to detect a person at that distance, but what is more likely is that the spatial mapping of the Hololens was unable to detect a surface that far away to render the distance hologram on.

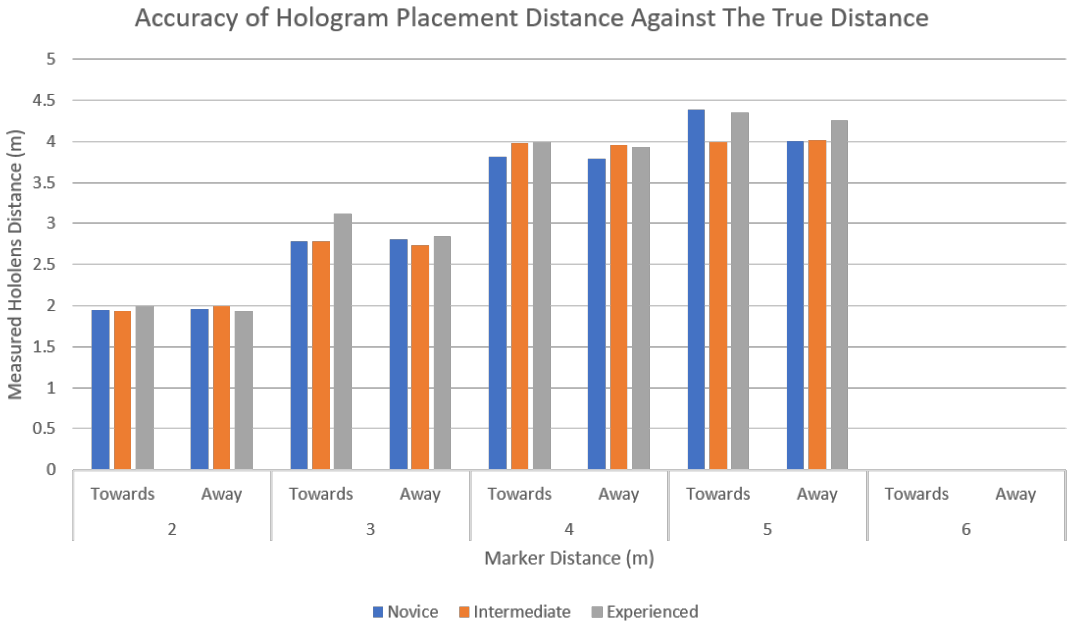
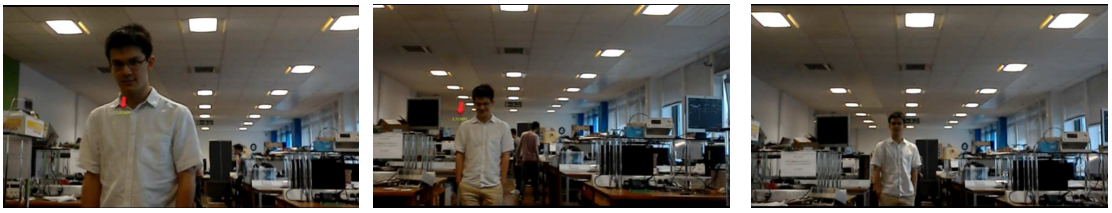


Figure 6.2: The graph shows that there is no significant difference between the orientation of the target. We note the decrease in accuracy past 4m, and the lack of holograms at 6m.

The data shows that there is no significant difference in hologram placement distance for target persons facing towards or away from the Hololens. However, we note that the hologram placement accuracy decreases the further away the target person is from the Hololens. This is further shown from the Hololens view recordings in Figure 6.3, whereby we can see that the placement of the hologram is not on the target person, but a surface just behind the target.



(a) Subject at 2m (b) Subject at 3m (c) Subject at 6m

Figure 6.3: Target person at different distances viewed through the front-camera of the Hololens. We note that the accuracy of the hologram placement decreases as the target is further away.

After the test, we asked the test subjects on qualitative feedback on how easy it is to view a hologram. As expected, the novice Hololens user had difficulty seeing holograms due to the limited FOV of the device. The novice also stated that the holograms were being rendered too high up, as they could only see the bottom of the hologram, as the rest of it was slightly out of view unless they tilted their head upwards. The intermediate user had a similar experience, but only had difficulty in seeing the holograms when the target person was closer than 2 meters. However, all three users complained about the limited FOV of the device and how it reduced their view of the surroundings.

6.1.5 Discussion

From our results, we note that the test subjects were unable to see a hologram when the target person was more than 5m away, which indicates that there is a limitation with the spatial mapping of the Hololens. We suspect that this is due to a surface mesh not being rendered for the detected person, since the spatial mapping class was unable to discern between the person and the background at that distance. Secondly, we bring up the issue of hologram placement accuracy compared to the real world position of the target person. The graph visualizes this issue in Figure 6.2, which shows a drop in accuracy for distances beyond 4 meters. Furthermore, Figure 6.3.b shows the hologram being rendered on a surface behind the target person.

We believe that these issues are caused by ray casting through an incorrect world coordinate. Firstly, the frames captured by the front-facing camera have to be compressed and streamed to a partner PC. The HDD system then processes the frames before being streamed back across the network. The culmination of the time spent being processed and streamed means that there is a slight delay between the detection of the person and their current position as interpreted by the Hololens. As such, when the Hololens unprojects the image coordinate of the detection to obtain a world coordinate, the placement of the initial GameObject is slightly incorrect. The system uses the ray casting function to pass a ray through the current position and expects the spatial mapping to detect a surface behind the point corresponding to the detected person. However, due to the slight delay, the ray continues to travel until it hits the spatial mapping mesh of the background instead of the mesh of the person. This explains why holograms are correctly placed on detections which are nearer, since the person is larger in the frame, and as such, there is more surface mesh area for the ray cast to hit.

6.2 Gazebo Simulation

Before testing the system using ARTA in the real world, we used the robot simulation tool Gazebo to test how the reactive control system would manipulate the ARTA control signals in response to person detections at different distances. The purpose of this simulation is to test the communication between ARTA and the Hololens, and how the reactive control system manipulates the velocity of the wheelchair as it approaches a detected person. We also want to test if the system can decide if the detections are to the side of the wheelchair or in front, and react appropriately.

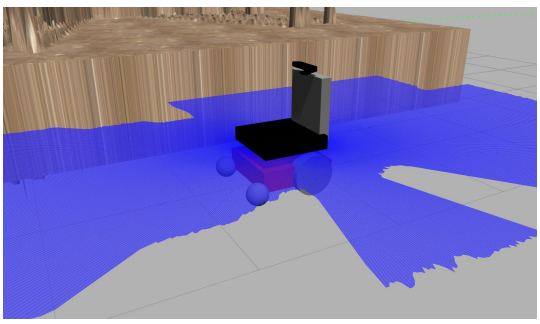
6.2.1 Test System Description

For this test, we use the same system like the one described in Section 6.1.1, with the addition of the Gazebo simulation of the ARTA powered wheelchair. We communicate the ARTA control signals to the Hololens, which checks the trajectory of the wheelchair and determines if a collision with a detected person is imminent. If the reactive control system decides a detected person is a collision risk, it reduces the velocity of the wheelchair to avoid a collision.

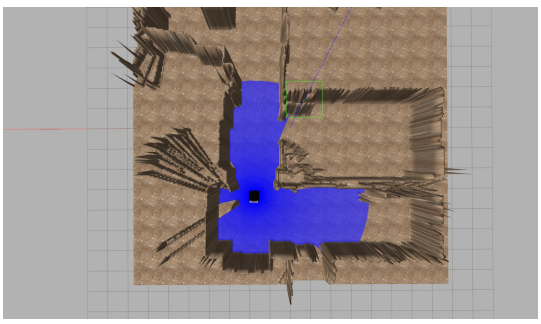
6.2.2 Test Setup

Gazebo Simulation

We use the Gazebo Simulation software to load a world built using the height map for the hallway outside the Personal Robotics Lab on the 10th floor. We then load the Unified Robot Description Format (URDF) model of ARTA into the world and run the ROS nodes controlling the powered wheelchair. We control the simulated wheelchair using the keyboard on the computer instead of a joystick.



(a) URDF model of ARTA with joints that model the kinematics of the wheelchair.

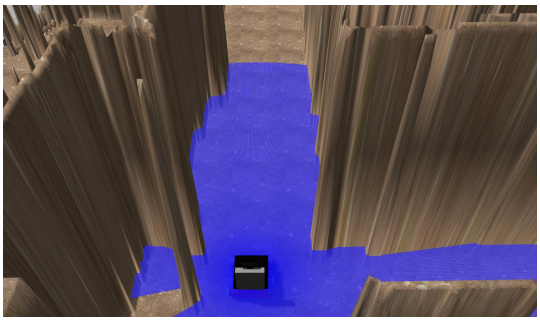


(b) An aerial view of the Lab Hallway on the 10th floor.

Figure 6.4: The Gazebo Simulation of the lab hallway outside the PRL. The blue mesh represents the range of the simulated laser scanner.

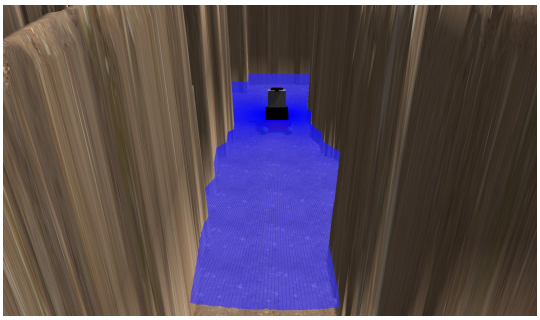
Real World Lab Setup

We are trying to test how the velocity of the wheelchair changes when the system detects a collision risk. We model the movement of the wheelchair in the Gazebo simula-



(a) Gazebo Simulation of hallway from behind the model.

(b) Corresponding real world lab setup from behind.



(c) Gazebo Simulation of hallway from in front of the model.

(d) Target persons will stand on the markings in front of the test subject which are 1m apart.

Figure 6.5: We emulate the real world position of the Gazebo model position in the ICRS lab. We monitor the simulated velocity of the wheelchair as we vary the distance the target person is from the Hololens.

tion, but we need real human detections to test the reactive control system. We use the same setup used in Section 6.1.2, where we use the long hall between benches in the 5th floor ICRS lab to replicate the hallway outside the PRL lab.

Monitoring ROS Topics

We record the ROS topics publishing the linear and angular velocities of the simulated wheelchair using a ROS bag. We monitor the following topics:

- `/navigation/main_js_cmd_vel`, the velocity controlled by joystick inputs.
- `/holo/cmd_vel`, the reactive control velocity.
- `/arta/cmd_vel`, the simulated velocity of the wheelchair.

6.2.3 Test Procedure

We asked the test subject to sit in a chair in the middle of the hallway wearing the Hololens. Similar to the experiment in Section 6.1.2, the target person stands at different distances to the test subject. Using the keyboard, we control the simulated wheelchair driving it forward down the simulated hallway. At each marked distance, we record the joystick, reactive control, and final velocities of the simulated wheelchair using the

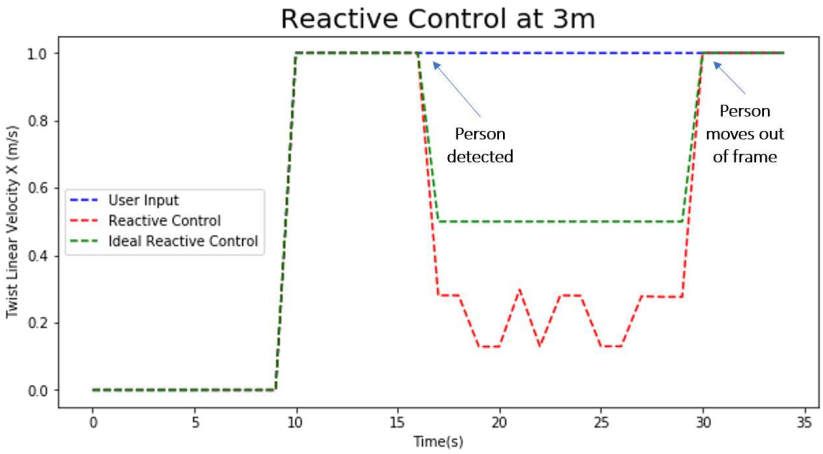
ROS bag for comparison. We test distances between 1 and 4 meters away since the reactive control system only takes over for detections closer than 3 meters.

After the distance tests, the test subjects were asked to look to the side as the simulated wheelchair moved forward. We asked the target person to stand to the side of the wheelchair to provide a human detection. This was done to test if the system can realize the detection is not in the trajectory of the wheelchair, and for the system to realize reactive control is not necessary.

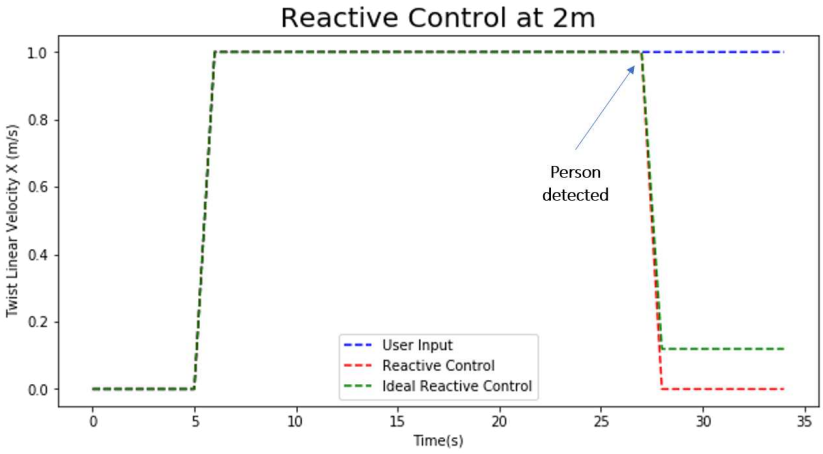
6.2.4 Results

We present the results of this experiment in Figure 6.6. We note at 3 meters that the reactive control system detects a collision risk and reacts appropriately by reducing the velocity of the wheelchair. We see in Figure 6.6.a that the output velocity of the reactive control system is below the ideal reactive control velocity of 0.5m/s. Furthermore, the velocity oscillates between 0.3m/s and 0.1m/s, well below the ideal velocity. Figure 6.6.b shows a similar trend, whereby the output velocity of the reactive control system is 0m/s instead of the ideal value of approximately 0.1m/s in the ideal case. Similarly, Figure 6.6.c also shows a discrepancy. More significantly is that we see oscillations in the velocity caused by flickering detections. At a distance of 4 meters, the reactive control was not activated. As such, there is no difference between the joystick input and reactive control velocities, so we omit those results from the figure.

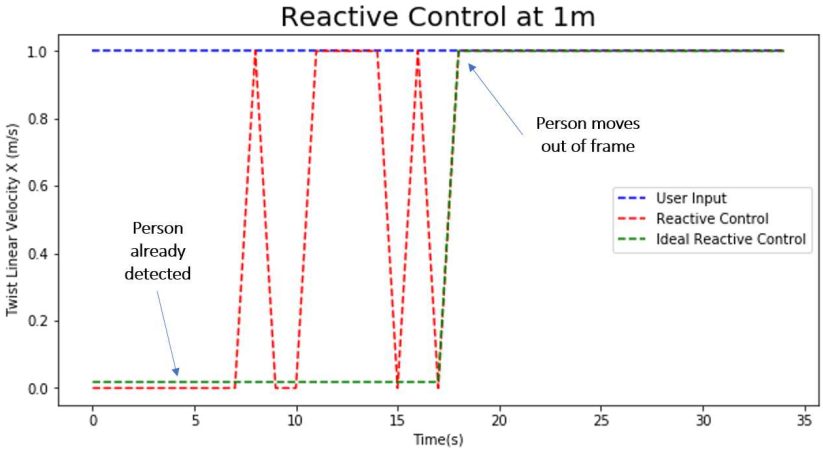
With respect to the head-turning results, we noted that the system was always able to recognize when the test subject was looking forward or to the side, and we saw no change to the joystick inputs when the user was not looking ahead. This indicates that the reactive control system is not activated as intended.



(a) Upon detection, the reactive control reduces the velocity to around 0.3m/s. When the collision risk is gone, it returns to 1.0m/s.



(b) At a distance of 2m from the detection, the reactive control system prevents the wheelchair from moving at all.



(c) The reactive control output velocity oscillates between 0 and 1 due to flickering detections. HDD fails to detect people if they are too close.

Figure 6.6: Reactive Control System velocity outputs at different distances.

6.2.5 Discussion

Due to the issues with hologram placement accuracy discussed in Section 6.1.5, we can explain the discrepancy between the ideal reactive control velocity and the simulated reactive control. As was discussed in the previous section, the GameObjects and holograms are not always placed in the correct position or on the correct surface. As such, the distance between the PWU and the GameObject in the Unity world varies slightly compared to the real world measurement. When holograms are placed slightly closer to the PWU, the reactive control system responds by reducing the velocity more than the ideal case, as seen in Figure 6.6.

We also wish to discuss the oscillations in the velocities set by the reactive control system. For the graph in Figure 6.6.a, the velocity oscillates within a small range below the ideal reactive control velocity. These small oscillations are most likely caused by flickering holograms. The holograms are ray cast onto the surface of the detected person, however, due to issues with hologram stability at a distance, the placement of the hologram and GameObject oscillates between two values. This is rendered to the user as a hologram flickering between two positions on the detected person. As such, the distance between the target and the PWU changes, causing oscillation in the velocity output of the reactive control system.

With regards to the larger oscillations seen in Figure 6.6.c, we believe that this is caused by the failure of the HDD system to detect a person standing that close to the PWU. However, we are unsure of the root cause of this problem. There is a possibility that this issue is caused by the YOLO detector being unable to detect a person when only their torso and head are visible. The more likely cause is that the OpenPose keypoint estimation fails due to it being unable to estimate the key points due to the person being too close. As part of our optimization to run both the tracker and pose estimator on the limited GPU memory, we reduced the network resolution of OpenPose, which results in less accurate key point estimations. The reduction in network resolution may also explain why the HDD system is unable to detect people further away, since the pose estimator fails for persons far away, as seen in Section 5.1.5.

Finally, we discuss the Hololens and ARTA frame alignment to determine if a detected person is in front of the wheelchair. From the tests, we noticed no change in the forward velocity of the simulated wheelchair. Furthermore, the cursor rendered on the holographic screens changes to the colour red when the PWU is not looking forward, and changes to green when the user is facing forward. This shows that the calibration process and alignment was carried out successfully. To conclude this test, we have found that the reactive control system works in simulation, giving us the confidence to test the full system in the real world.

6.3 Full System

As a final test, we want to see the whole system in action with all three devices functioning and communicating with each other. We want to check whether the HDD system can detect people who are walking towards the PWU operating ARTA. We also want to see how the reactive control system responds to collision risks in a real-world setting, and whether the response is fast enough to compensate for people walking and the wheelchair moving forward. This test utilizes ARTA, the HDD system, and the Hololens running the full Unity application.

6.3.1 Test Setup

This test was conducted in the hallway outside the Personal Robotics Lab on the 10th floor of the EEE building. The reason for this choice of test location is the pre-existing height maps for localisation and navigation that have been developed by the members of the PRL. Furthermore, this experiment was designed to test the real world application of the Gazebo Simulation test we conducted in Section 6.2.

This experiment involves positioning ARTA at the end of the hallway. The test subject is sat in the wheelchair wearing the Hololens running the Unity application. Communication between the Hololens and the HDD system is established and checked to ensure the HDD system is receiving video frames and processing them. We then check the Hololens for detections by asking the test subject to confirm they can see holograms being rendered. We then ensure ARTA is communicating with the Hololens by trying to move the wheelchair without any detections and checking the `/holo/cmd_vel/` ROS topic for messages.

6.3.2 Test Procedure

To test the effectiveness of the system, we designed three different scenarios:

1. Target Away: ARTA and PWU driving forward in the same direction as the target person walking down the hallway.
2. Target Towards: ARTA and PWU driving forward as the target person walks towards the PWU.
3. Looking Away: ARTA and PWU driving forward, PWU is looking to the side so that the target person is in the FOV of the Hololens but not in the forward trajectory of ARTA.

These scenarios were designed to test whether the Hololens will detect a person as a collision risk and translate it to a reactive control output that slows down the powered wheelchair, preventing a collision. We illustrate the positions and motions of the target person using the Gazebo simulation map in Figure 6.7 since it provides an aerial representation of the hallway.

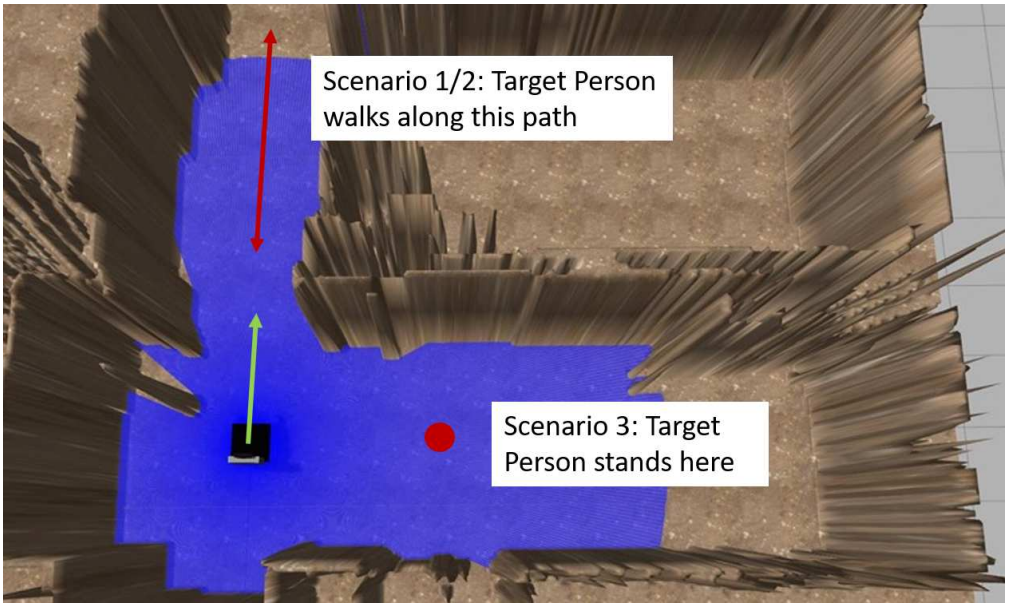


Figure 6.7: The experimental setup used to test the full system. We ask the target person to walk down the hallway for Scenarios 1 & 2. For Scenario 3, the target person stands still.

Target Away In this scenario, we are testing the system's ability to detect a person walking in the same direction as the wheelchair. We want to test the system response to the detection which we expect to be a reduction in the forward velocity of ARTA, allowing the robot to continue moving forward, albeit at a slower velocity. We also want to evaluate how the system responds to the person moving out of the frame.

Target Towards This experiment was designed to test how the system responds to a collision risk that is approaching the wheelchair. The target person is asked to walk towards the wheelchair as the wheelchair moves forward. We are testing for the system to detect the person and to rapidly decrease the forward velocity of the wheelchair as the target person comes closer. We expect the system to stop moving completely when the target person is closer than 2m away.

Looking Away We ask the test subject wearing the Hololens to drive ARTA in a forward direction while looking to the side. The target person is asked to stand to the side of the PWU, 2m away from the start position of ARTA. This scenario is designed to test if the system can recognize that the detected person is not in the current trajectory of the wheelchair, and so the reactive control system does not need to modify the user joystick input commands to avoid a collision.

During Scenarios 1 & 2, we asked the Target Person to maintain a constant walking speed slightly faster than the top speed of 1.0m/s of the powered wheelchair. We also asked the Target Person to continue walking unless they were certain that a collision

would occur, in which case we instructed the target person to avoid the collision themselves.

(a) Scenario 1: Target person walking in the same direction as forward motion of ARTA.

(b) Scenario 2: Target person walking towards the PWU and ARTA.

Figure 6.8: The path taken by the Target Person along the hallway outside the PRL.

6.3.3 Results

We measured the success of a scenario by whether human intervention was required to prevent a collision. We define human intervention as the act of:

- The PWU stopping the forward joystick input manually or by navigating the wheelchair out of the way to prevent a collision.
- The Target Person having to move off the assigned path to avoid a collision.

Target Away

Through our observation of this scenario, we noticed several things:

- System was unable to detect the target person beyond 3-4 meters in front of the wheelchair.
- ARTA moved much slower than expected despite the target being between 2-3m in front of the device.

We define a detection as the placement of a hologram at the position of the detected object. The test subject was unable to see a hologram of the green arrow on the surface of the target person when the target person was standing further than 3-4 meters away. The PWU and ARTA retained a relatively smooth velocity behind the target person walking away, and no collision with the target occurred. When the target person left the FOV of the PWU, the wheelchair returned to the top velocity of 1m/s.

Target Towards

In this scenario, the PWU and ARTA drove forwards as the target person walked towards the powered wheelchair. We noticed that as soon as the target person was approximately 3m away from the PWU, the velocity of ARTA reduced rapidly in a sudden

jerking motion. As the target person got closer, we noted that ARTA oscillated between moving forward and remaining stationary, instead of remaining completely still. When the target person was very close to the PWU, ARTA began moving forward, indicating the reactive control system had failed to notice a detection and the system required human intervention.

Looking Away

This test involved driving the wheelchair forward while the PWU looked at a target to the side. For this scenario, we noted no change in the velocity as the PWU detected the target person, which indicates that the reactive control system was not activated as expected.

Summary of Scenarios

To summarise the results of the scenarios, we want to highlight several points we noticed during the test. Firstly, we want to highlight that the system can determine whether a detection is in front of the powered wheelchair and in the way of the current trajectory. However, occasionally, the system is unable to detect target persons in front of the wheelchair. This issue was observed when the target was either far away from the PWU or extremely close. Secondly, we note that the reactive control causes a sudden and much more rapid decrease in velocity than we intended. Finally, we want to highlight the issue the system faced when the target person was very close, which caused Scenario 2 to require human intervention.

6.3.4 Discussion

We now discuss the issues highlighted in the previous section. We begin with the issue whereby the system fails to detect a target person standing very close to the PWU. As discussed in Section 6.1.5 and Section 6.2.5, this is most likely due to a failed detection by the HDD system for people too close to the PWU. This event is similar to the oscillation issue we faced in Section 6.2.5, where for close detections, the velocity output of the reactive control system oscillated between 0m/s and 1m/s.

Furthermore, due to the variance in hologram placement distance, we can explain the rapid decrease in velocity output at 3m that occurred in Scenario 2. We believe that as the target person crossed the 3m distance and activated the reactive control system, the GameObject representing the person was placed slightly in front of the target. This incorrect placement caused a sudden drop in velocity. Also, as the distance between the PWU and the target decreased, the ray casting of the GameObject had to correct itself multiple times, causing the distance between them to fluctuate. These fluctuations explain the rapid changes in velocity that caused ARTA to oscillate between movement and remaining stationary.

As discussed in Section 6.2.5, the lack of detections for people far away may be caused by incorrect pose estimations by the OpenPose network in the HDD system due to the low network resolution.

6.3.5 Overall System Discussion

By testing the full system, we were able to assess how our implementation would perform in a real-world scenario. Although the real world scenario was simplified to only three controlled situations, we immediately noticed that there is a lot of room for improvement. First and foremost is in the HDD system, where persons far away were not recognized. From our tests, we established a possible limiting factor being the resolution of the OpenPose network, which may fail key point estimation for small figures in the distance. Secondly, in the Hololens Unity application, where we convert the image coordinate of the detection to a world coordinate. Here we rely on the library provided by Vulcan Technologies, which utilizes the camera parameters to form an un-projection matrix to obtain the world position of the pixel. Due to the delay from the streaming to a partner PC and processing through the HDD system, the world coordinate is slightly off and not representative of the world position of the person in the current frame. Finally, the rapid decrease in ARTA forward velocity when a detection enters the reactive control activation range. Due to the variance in hologram distance, we realized it was unsuitable to use a scaling function that reduced the speed of the wheelchair so aggressively. We discuss these issues further in the next chapter of the report.

Chapter 7

Evaluation

This chapter summarises the capabilities of the augmented reality system as a product against the requirement capture, an evaluation of the techniques used to produce the system and the achievements of the project compared to the goals set in the interim report.

In the interim report, we proposed the "Human Detection System," which would estimate the trajectories of people utilizing a moving object detector. Instead, the final product implements a Human Detection & Direction system that relies on object detection and body pose estimation. The first and most challenging step of the project was developing the video stream of the front-facing camera on the Hololens. Due to the reliance of the HDD system on the visual input, this made the video stream a critical part of the project. As such, we considered multiple approaches to achieve a real-time stream with a relatively high frame rate. Using Unity, we were able to access the front-facing camera and stream the captured frames to a partner PC. From there, the HDD system can use the YOLO object detector to detect humans in the captured frames and discern the direction they are walking in using the OpenPose pose estimation network and object tracking. As such, we have achieved the requirements set in Section 3.2.

In spite of our achievements, we must be critical of the system we have implemented. Through our development, we found it was challenging to develop a video stream of the front-facing camera due to the limitations of the Unity game engine. We discuss the issues of compression in Section 5.2.2, and how it limits the user experience to 5 FPS on the holographic lenses of the Hololens. Despite discussing potential solutions with the members of the PRL and open source contributors online, we were unable to find a realistic solution for faster image compression that could be run outside of the Unity main thread. As such, we had to continue with the project, knowing that there was a limitation on the user experience in terms of holographic stability. Furthermore, due to the limitations of the front facing camera (which captures at 30 FPS), image compression and transfer over the network, we can only achieve a modest 10 FPS on the partner PC.

With regards to the person detector in the HDD system, we were able to improve the detections of partially occluded persons and smaller figures at a distance compared to the pre-trained model. We verified this through our visual testing on the MOT dataset

and our recorded videos around Imperial College London. On the other hand, we did not end up using the object tracking capabilities of Deep SORT to its full potential. We initially proposed the use of the tracker to determine the directions using only a 2D image by tracking the motions of objects across frames. However, we found this task difficult when the detected person was walking towards or away from the camera, as discussed in Section 4.2.2. However, instead of removing the tracker, we kept it in the system for potential future work to increase the hologram stability and better direction estimation.

Furthermore, the decision to use body pose estimation to determine the direction a person is walking in was an adventurous one. A much simpler option would have been to rely on object tracking and using the depth camera on the Hololens to see how the position of the object changed in the real world. This would have reduced the GPU memory usage, the complexity of the HDD system, and the amount of work required to get the network to run on ROS. In hindsight, we realize it was misguided to rely solely on computer vision techniques to determine the human direction and that a much safer option would have been to utilize the sensors for depth perception available on the Hololens and ARTA.

Also, in the interim report, we proposed the use of the Pupil Labs eye tracker as a form of wheelchair control. However, streaming the captured video frames caused a limitation in the frame rate of the user experience, and as such, we were unable to implement this goal. We discuss the full reasoning for this in Section 4.4.3. We also proposed the development of Hologram warnings using the eye tracker to alert the user on a potential collision. In the final implementation, we have developed hologram visualizations of the directions detected people are walking in which act as a warning for potential collisions. We achieve this through the detections produced by the HDD system and create a map of the detections by utilizing the spatial mapping. Since one of the goals set in the interim report was to control the powered wheelchair using the augmented reality headset, we implemented the reactive control system across ARTA and the Hololens. However, from our testing, we found that there are several issues with the reactive control system, and most of them stem from the fact that we are using a single video input as our source of decision making.

From our results, we realize that the combination of a low frame rate from the video stream and the network delay during transfer between the Hololens and partner PC results in latency between the processed frame and current view of the PWU. This latency also results in incorrect projections of image pixels to world coordinate space. As discussed in Section 6.3.4, this causes holograms to jitter in position, as well as affecting the output of the reactive control system velocity. To avoid the issue of inaccurate hologram placement in the world, a solution would have been to use Spatial Anchors and placing holograms relative to the anchors. This would also increase the accuracy of the hologram in the real world relative to the virtual world rendered by the Hololens. However, we did not use spatial anchors because one of the ideas for this system is for PWU to be able to wear it in the streets, in unmapped places where spatial anchors are not already placed.

Finally, we discuss the reactive control system, which we implement on the Hololens and ARTA. The system relies solely on the mapping of detected people created by the Unity application. However, we found that this results in a system that can be unstable, depending on the accuracy of the detections. From our tests, we also noticed that the lighting of the room affected the systems ability to detect people, due to the resolution of the front-facing camera. As such, a better approach for collision avoidance would have been to utilize the PRL developed obstacle avoidance ROS packages built for ARTA. These libraries would have made the development of the reactive control system much simpler. Furthermore, from our tests, we also noticed that the wheelchair would reduce its velocity rapidly when detecting an object moving towards it. This is due to the choice of scaling function, which are much too aggressive for real-world use. Instead, a smoother scaling function that reduces the velocity slowly is more suitable for a powered wheelchair, resulting in a smoother ride.

Chapter 8

Conclusion and Further Work

8.1 Conclusion

Throughout this report, we have discussed the implementation of an augmented reality system to aid PWUs in navigating areas with people as potential collision risks. We outlined the requirements for the system to handle this scenario in the Requirement Capture section of the report. From our tests, we can show that we have implemented a working proof of concept, using the Microsoft Hololens as the augmented reality device, and front-facing camera as the visual input for people detection and analysis.

One of the contributions of this project is the development of a Unity application that streams the front-facing camera of the Hololens to a partner PC. The ability to process the frames on a computer with a GPU opens up a whole avenue of research for computer vision techniques using the Hololens. This report explores the use of these techniques in the Human Detection & Direction system, which uses the YOLO object detector, Deep SORT tracker, and OpenPose body pose estimation networks. To make this accessible to future researchers, we make available the Unity framework, which implements the video streaming across ROS topics. However, this report also highlights the limitations of using the Unity engine for video streaming, and suggest that any future researchers should develop a UWP application that accesses the Hololens video stream, to make use of the multi-threading capabilities of the device.

Secondly, we contribute a version of Darknet that has been modified to run as a ROS node, allowing seamless integration with other ROS packages. We achieved this by wrapping the core Darknet libraries using Python and providing an interface between the neural network framework and ROS topics to pass messages containing the images and bounding box detections. In addition to this, we have also trained the YOLO object detector using the CrowdHuman dataset. We provide the pre-trained weights that can be run on the ROS Darknet framework. We also explored the use of body pose estimation as a method of determining the direction a detected individual is walking in. We have developed Python wrappers around the network to allow it to accept ROS topic messages as input and a way for it to output its keypoint estimations.

Thirdly, in addition to the video streaming capabilities of the Unity application, we have also developed an augmented reality experience for PWU which visualizes the direction people are walking in, based on the results of the HDD system. These holograms are rendered by the Hololens and are used as visual aids by the PWU to avoid collisions. Due to the frame rate limitations of the application, the holographic visualizations are not completely stable but provide a reasonable indicator of direction to the PWU.

Finally, we have developed a simple collision avoidance system that utilizes the spatial mapping capabilities of the Hololens to determine the distance between the PWU and the detected people. The reactive control system monitors the input velocity commands sent by the PWU using the joystick on ARTA and selectively controls the final velocity of the wheelchair when it detects a collision.

8.2 Future Work

8.2.1 Utilizing ARTA Sensors

To achieve a more accurate mapping of the surroundings, the system should not rely on a single sensor input in the form of the front-facing camera. Instead, the visual input and positions of the object obtained from the front-facing camera should be used in conjunction with other sensors, such as the laser scanners attached to the base of ARTA. Furthermore, the PRL has already developed ROS packages that implement this on ARTA. The future work would involve comparing the positions of the detected persons in the Hololens frame and ARTA frame to estimate a better world position.

8.2.2 GameObject Tracking in Unity

In the current Unity application, we create a new GameObject representing a detected person and delete it every frame. We do this since we determine the direction a person is walking in from the output of the OpenPose network. As such, the tracker ID is not used beyond visualization.

An alternative approach would be to create a single GameObject for each tracking ID. These GameObjects are then compared across frames, and if the tracker ID exists in the new frame, the previously instantiated GameObject is rendered. When the tracker ID is not in the frame, we set the mesh of the hologram to transparent, and the hologram is not visible to the PWU. This would allow the Unity application to keep track of the positions of the detected objects across frames, and reduce the error in the accuracy of hologram placement. This would also mean that the Hololens was aware of detected persons, even when the user turns their head to the side and the objects are no longer in the FOV of the front-facing camera.

8.2.3 Advanced Reactive Control

Rather than have the reactive control system reduce the speed of ARTA when it detects a collision risk, a more advanced technique would be a system that takes over and navigates the wheelchair out of the way of the detected person. This system would be implemented using either the additional ARTA sensors or GameObject tracking we mentioned in the previous sections as a way of keeping a better mapping of the surroundings of the Hololens. We could then use the object avoidance libraries developed by the PRL to control ARTA when it detects objects, and calculate the best path that avoids a collision.

8.2.4 Video Streaming

The major limitation of the Unity Game engine is the fact that image compression can only be done in the main thread of the application. This limits the frame rate the scene can be displayed at, resulting in latency when rendering holograms in the surroundings. This was why we chose to instantiate and delete holograms for each detection instead of having them persist across frames.

To solve this problem, one would have to develop a way of compressing the images outside of the main thread. This is a difficult problem to solve since Unity does not allow access to textures and GameObjects outside of the main thread due to thread safety issues. As such, an alternative approach would be to somehow access the video streams outside of Unity, perhaps using the HololensForCV library provided by Microsoft, to stream the front facing camera to a partner PC. The partner PC would then stream the captured images back to the Hololens and the detections, into the Unity application. This, however, would involve writing a UWP application that could run in the background of the Hololens, as well as using Windows networking protocols to stream the video data from the Hololens.

Bibliography

[1] M. Zolotas, J. Elsdon, and Y. Demiris. Head-mounted augmented reality for explainable robotic wheelchair assistance. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1823–1829, Oct 2018.

[2] Rodrigo Chacón-Quesada and Yiannis Demiris. Augmented Reality Control of Smart Wheelchair Using Eye-Gaze-Enabled Selection of Affordances. pages 1–4, 2018.

[3] Dahlia Kairy, Paula W. Rushton, Philippe Archambault, Evelina Pituch, Caryne Torkia, Anas El Fathi, Paula Stone, François Routhier, Robert Forget, Louise Demers, Joelle Pineau, and Richard Gourdeau. Exploring powered wheelchair users and their caregivers' perspectives on potential intelligent power wheelchair use: A qualitative study. *International Journal of Environmental Research and Public Health*, 11(2):2244–2261, 2014.

[4] Y. Hou and G. K. H. Pang. Human detection in crowded scenes. In *2010 IEEE International Conference on Image Processing*, pages 721–724, Sep. 2010.

[5] Dongdong Zeng, Ming Zhu, Tongxue Zhou, Fang Xu, and Hang Yang. Robust background subtraction via the local similarity statistical descriptor. 2017.

[6] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[7] M. Piccardi. Background subtraction techniques: a review. In *2004 IEEE International Conference on Systems, Man and Cybernetics (IEEE Cat. No.04CH37583)*, volume 4, pages 3099–3104 vol.4, Oct 2004.

[8] M. Hirabayashi, S. Kato, M. Edahiro, K. Takeda, T. Kawano, and S. Mita. Gpu implementations of object detection using hog features and deformable models. In *2013 IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*, pages 106–111, Aug 2013.

[9] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–I, Dec 2001.

[10] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[11] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, June 2014.

[12] C. Dicle, O. I. Camps, and M. Sznaier. The way they move: Tracking multiple targets with similar appearance. In *2013 IEEE International Conference on Computer Vision*, pages 2304–2311, Dec 2013.

[13] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *Proceedings - International Conference on Image Processing, ICIP*, volume 2016-Augus, pages 3464–3468, 2016.

[14] R. E. Kalman and R. S. Bucy. New Results in Linear Filtering and Prediction Theory. *Journal of Basic Engineering*, 83(1):95, 1961.

[15] N. Wojke, A. Bewley, and D. Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649, Sep. 2017.

[16] Roberto Valenti, Nicu Sebe, and Theo Gevers. Combining head pose and eye location information for gaze estimation. *IEEE Transactions on Image Processing*, 21(2):802–815, 2012.

[17] E. Murphy-Chutorian and M. M. Trivedi. Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):607–626, April 2009.

[18] Vahid Kazemi and Josephine Sullivan. One Millisecond Face Alignment with an Ensemble of Regression Trees. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1867–1874, 2014.

[19] George Papandreou, Tyler Zhu, Nori Kanazawa, Alexander Toshev, Jonathan Tompson, Chris Bregler, and Kevin Murphy. Towards accurate multi-person pose estimation in the wild. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-January, pages 3711–3719, Jan 2017.

[20] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2017.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[22] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter V. Gehler, and Bernt Schiele. Deepcut: Joint subset partition and labeling for multi person pose estimation. *CoRR*, abs/1511.06645, 2015.

[23] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *CoRR*, abs/1812.08008, 2018.

[24] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part i. *IEEE Robotics Automation Magazine*, 13(2):99–110, June 2006.

[25] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, Dec 2016.

[26] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, Jun 2017.

[27] D. Nister. A minimal solution to the generalised 3-point pose problem. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, June 2004.

[28] Paul Milgram and Fumio Kishino. A taxonomy of mixed reality visual displays. *IEICE Trans. Information Systems*, vol. E77-D, no. 12:1321–1329, 12 1994.

[29] Microsoft. Windows Mixed Reality Documentation, 2018.

[30] Microsoft. Microsoft HoloLens HoloLens Device Specifications. 2015.

[31] Guanghan Ning, Ping Liu, Xiaochuan Fan, and Chi Zhang. A top-down approach to articulated human pose estimation and tracking. *CoRR*, abs/1901.07680, 2019.

[32] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. *CoRR*, abs/1605.03170, 2016.

[33] Ross Girshick, Ilija Radosavovic, Georgia Gkioxari, Piotr Dollár, and Kaiming He. Detectron. \url{https://github.com/facebookresearch/detectron}, 2018.

[34] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, June 2017.

[35] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *CoRR*, abs/1804.02767, 2018.

[36] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[37] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. Crowdhuman: A benchmark for detecting human in a crowd. *CoRR*, abs/1805.00123, 2018.

[38] Anton Milan, Laura Leal-Taixé, Ian D. Reid, Stefan Roth, and Konrad Schindler. MOT16: A benchmark for multi-object tracking. *CoRR*, abs/1603.00831, 2016.

[39] Stefan Leutenegger. Representations and Sensors. 2019.

[40] Massimiliano Patacchiola and Angelo Cangelosi. Head pose estimation in the wild using convolutional neural networks and adaptive gradient methods. *Pattern Recognition*, 71, 06 2017.

[41] Joseph Redmon. Darknet: Open Source Neural Networks in C. \url{http://pjreddie.com/darknet/}, 2013.

[42] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *CoRR*, abs/1408.5093, 2014.

Appendices

Appendix A

Software

A.1 Developed Software

All code developed in this project is available in their respective GitHub repositories. We have provided instructions in the `README.md` file of each repository on how to setup the code and how to run it.

A.1.1 Partner PC

The following applications were developed to run on Ubuntu 16.04 with 8GB DDR3 RAM and a GTX 1050 Ti. ROS Kinetic is also required, as well as Python 2, gcc and g++. For a more complete listing, please refer to the medium

Darknet CrowdHuman

Source: `https://github.com/alaksana96/darknet-crowdhuman`

This repository contains the Darknet version used in this project. We also provide the code required to convert the CrowdHuman dataset to the Darknet format for training the YOLO detector. This repository is also used as a submodule in the `fyp_yolo` repository.

ROS Darknet/YOLO

Source: `https://github.com/alaksana96/fyp_yolo`

The ROS wrapper for the Darknet neural network framework. This repository relies on the Darknet CrowdHuman repository, and clones it as a submodule. When cloning this repository, please remember to clone it recursively using the `--recurse-submodules` flag

Yet Another CrowdHuman Tracker

Source: `https://github.com/alaksana96/fyp_yact`

Dependencies:

- `https://github.com/CMU-Perceptual-Computing-Lab/openpose`
- `https://github.com/alaksana96/deep_sort`

YACHT depends on the OpenPose network and a modified DeepSORT implementation. Please refer to the OpenPose repository for installation instructions. Furthermore, please clone the YACHT repository recursively using the `--recurse-submodules` flag.

A.1.2 Hololens

The following applications were developed using Unity for the Unity Engine running on the Microsoft Hololens.

Unity Application

Source: `https://github.com/alaksana96/fyp_hololens_unity`

This application requires Unity 2018.1.6f1 and Visual Studio 2017 Community edition to build and deploy to the Hololens.

A.2 Personal Robotics Lab Software

For the software developed by the Personal Robotics Lab, please email a member of the lab for access to the repositories:

- `https://github.com/ImperialCollegeLondon/arta`
- `https://github.com/ImperialCollegeLondon/prl_localisation`
- `https://github.com/alaksana96/prl_navigation`

We have modified the `prl_navigation` repository to work with our reactive control system, please email contact us on GitHub for access to the repository.