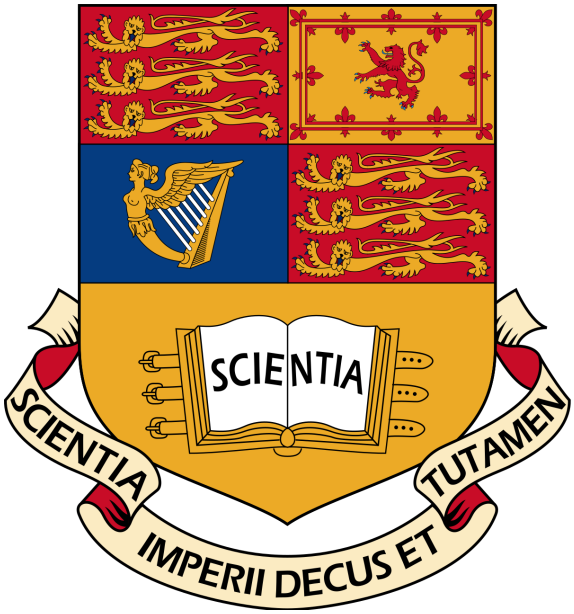
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2019



Project Title:	**Augmented Reality-assisted Human Robot Interaction**
Student:	**Aufar P. Laksana**
CID:	**01093575**
Course:	**EIE4**
Project Supervisor:	**Professor Yiannis Demiris**
Second Marker:	**Dr Tae-Kyun Kim**

Abstract Powered wheelchairs are becoming increasingly commonplace in the modern world. However, a major 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 utilizes computer vision techniques to predict the direction of travel of individuals in crowds, and implements an augmented reality system using the Microsoft Hololens that aids the PWU by displaying visual aids that indicate the motion of people. The system further aids the user by warning the user of potential collisions, allowing the PWU to make better navigation decisions. The project also explores the use of the system as a method of assitive control of the wheelchair, preventing collisions by stopping the powered wheelchair should the PWU not notice an individual crossing their path.

Acknowledgements This is for a few people

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 interest me and gave me the opportunity 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 5^{th} 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 everyday 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 give me the opportunity 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, thank you for being my brother. You have always supported me and 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.

Contents

Chapter 1

Introduction and Requirements

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

Chapter 2

Background

This project is focused 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. In order 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.

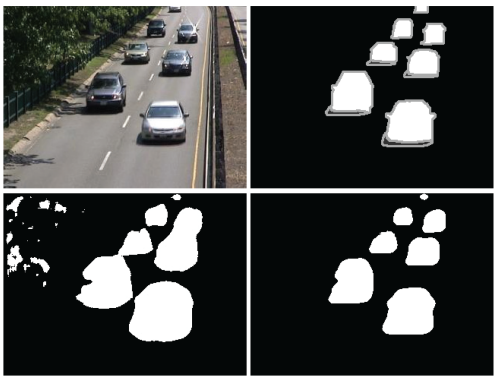
2.1.1 Direction of Research

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 identifiying parts of the body are not always visible.

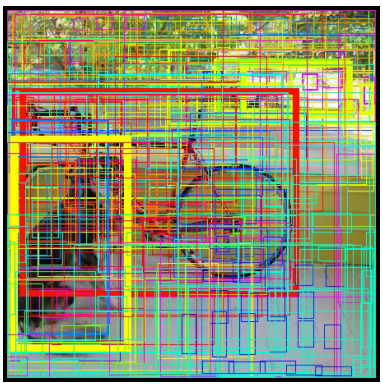
2.1.2 Review of Existing Methodologies

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 every single individual in a crowd impossible.

Despite the disadvantage, similar techniques can be used to detect humans in video. Most methods can be classified into two categories [1]. 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 can be classified into a human shape.



(a) Foreground Detection [2]



(b) Scanning Window [3]

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 [4]. 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 can then be 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 has the ability to 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 was detected in each window. However, it must be noted that this method is computationally expensive. In order to achieve real-time detection on a mobile robot, the use of a graphics processing unit (GPU) should be considered [5].

Classical Object Detection

Haar Cascades Haar cascades classifies images based on the value of simple features [6], 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 rapidly compute a small set of representative rectangular features.

A cascade of classifiers is then used to determine if the region is detected as 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 been 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 region. 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 [7], 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 approach 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.

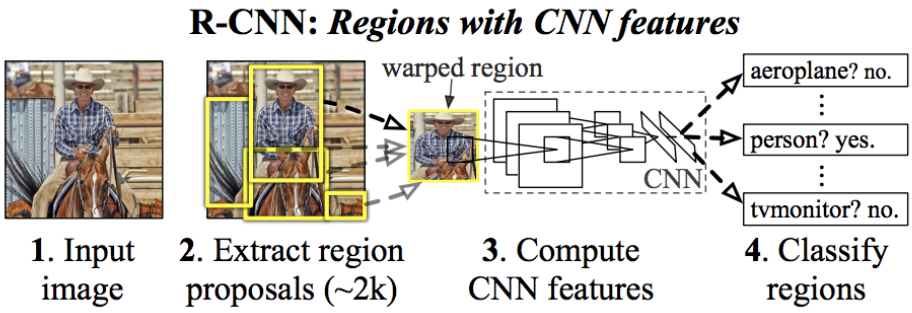


Figure 2.2: Visualization of the R-CNN approach. The number of region proposals contributes to the latency of this method.

R-CNN The R-CNN method uses a selective search to extract 2000 regions from an image [8]. The regions are selected by generating a large number of candidate regions and using a greedy algorithm to recursively combine similar regions into larger ones. 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 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 [3]. By looking at the image as a whole, the network can use features from the entire image to predict each bounding box.

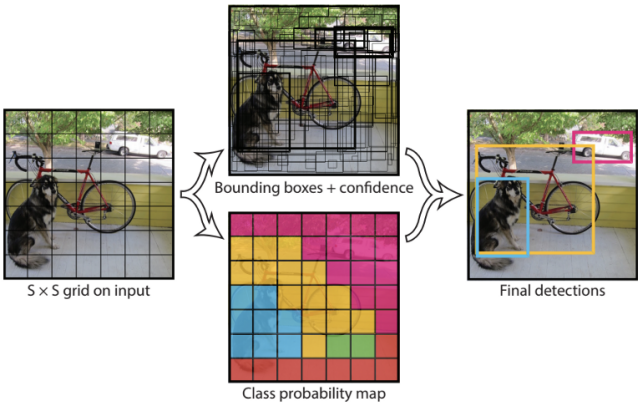


Figure 2.3: YOLO algorithm dividing a square image into grid-cells for bounding box prediction.

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 clearly 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 common 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 feedback to the PWU. This narrows our field of research to online object tracking techniques.

2.2.2 Review of Exisiting Methodologies

Pedestrian detection is often achieved by using a high quality object detector and associating the detections across frames [9]. The associations are based on the appearance and location similarity. Furthermore, it is possible to discern simple motion patterns from the 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 [10]. The technique estimates the inter-frame displacements of each detected objects with a linear constant velocity model. The state of each tracked object is modelled 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 [11]. 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 [12].

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 are 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 clearly see a trade-off between 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 [13]. 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 [14]. 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 [15]. 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 increase, 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 backwards. 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 [16] in building a network that utilizes the Faster-RCNN model as an object detector to obtain accurate bounding boxes of people in an image [17]. The key-points are then calculated using a a ResNet [18] 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 labelling an initial pool of body part candidates into subsets [19], is able to deal with an unknown number of people, and can infer that number by linking the part hypotheses.

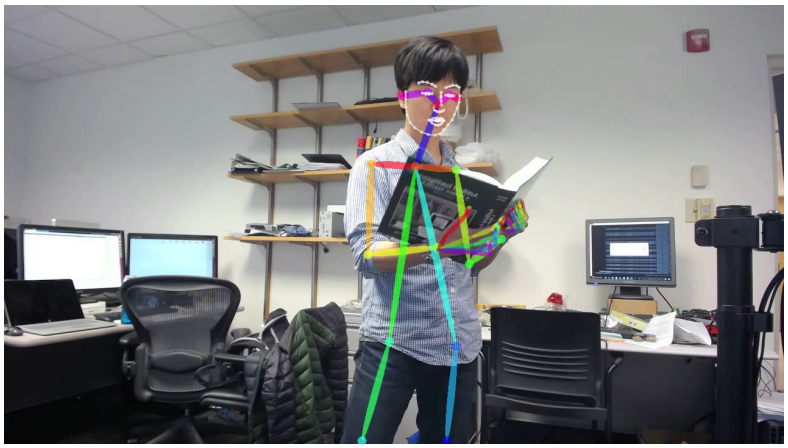


Figure 2.4: OpenPose body pose detection [20]. The network is able to determine an estimate of the key-points hidden by objects.

As such, OpenPose presents a method of multi-person pose estimation using a bottom-up approach [20]. The method 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. In order to help users navigate, the system must analyse the surroundings for potential dangers. As such, it is important to build up a thorough 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 on-line, without the need of an *a priori* knowledge of the robots location [21]. 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 [22], 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 estimate the position are the robot poses, the position of landmarks and the calibration parameters of the sensors.

In order 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 matter, 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 human objects walking around the wheelchair. This issue will be addressed in a later section.

Visual SLAM

Visual SLAM (vSLAM) is an implementation of SLAM that relies on visual inputs only. As stated in [23], 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 co-ordinate 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 co-ordinate 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 [24]. 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 [25], 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 [26]. 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 [27].

- 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 [28]. 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 [29].

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.

Figure 2.5: AR visualizations and markers for interaction [29]

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 main 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 encompasses 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 [29]. 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 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 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 visualization include arrows that indicate 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 had the ability to reactively control the device to prevent collisions with detected objects.

Features

- Receiving object detections in front of 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

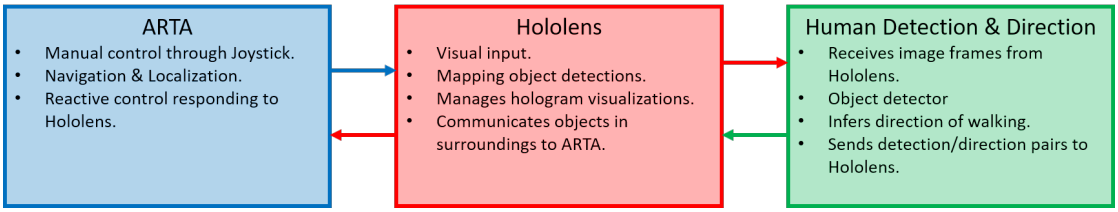


Figure 4.1: High level system diagram showing the flow of messages from ARTA and the HDD system through the intermediary device, the Hololens.

From a very high level view, we can map these requirements to the respective devices they will be operating on. The HDD system takes 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 hardware overall 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 consists 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 [10, 30]. In the case of top-down body pose estimation methods, detections can be the first point of failure [31]. 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 in 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 [29, 32, 33]. Further discussions with members of the Imperial Computer Vision & Learning Lab suggested the use of the YOLO object detector [3], due to its speed and having a lightweight implementation that could 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 [34].

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 [35].

Comparing Models To compare the accuracy of the bounding boxes produced by pre-trained model, sample videos were recorded using the Hololens and used as a base comparison point. It was quickly shown 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 people close together.

Pedestrian Dataset A common case 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 [36], which contains annotated images of people in crowded places. The annotations include bounding boxes for the

(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.

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 is able to 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 [12], while the pose estimator node uses the OpenPose [20] 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 [10] and Deep SORT [12].

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-labelled the objects with brand new tracking IDs.

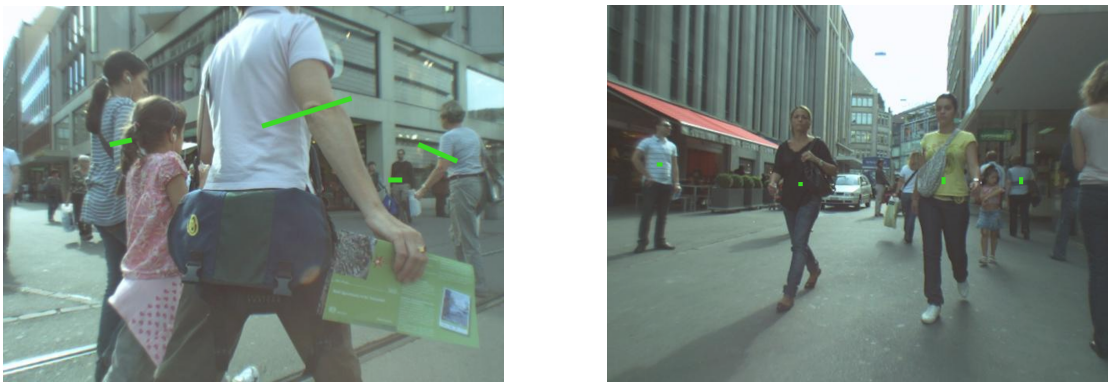
Deep SORT Deep SORT is an extension of the original SORT algorithm, but uses a deep network to generate feature descriptors for the predicted bounding boxes. We explain the algorithm in Section 2.2.2, but to repeat, instead of using the Intersection-over-Union association metric to compare bounding boxes, 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 this additional step reduced the problem of trackers being lost due to occlusion.

Figure 4.3: MOT16 benchmark [37] (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 co-ordinates 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 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 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.

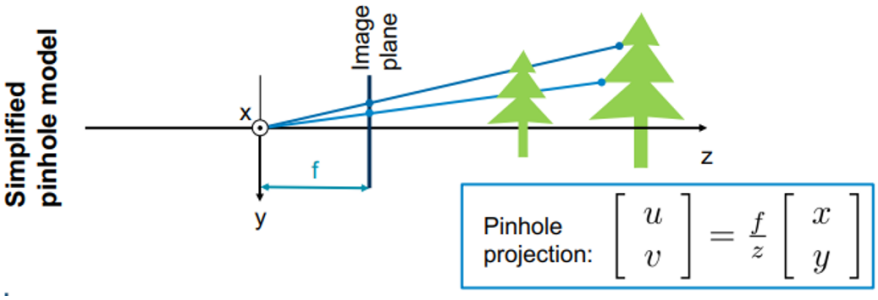


Figure 4.5: Pinhole Camera projection and the loss of the z-axis [38]. As such, it is not possible to determine the exact distance to an object without a depth camera.

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 [20].

Object Detectors & Bottom-Up Approaches The YOLO object detector outputs the original image and the associated bounding box co-ordinates 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.

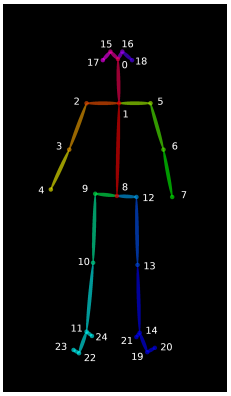


Figure 4.6: Keypoints produced by the BODY_25 model [20].

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 [39], since the library has a built in head-pose estimator.

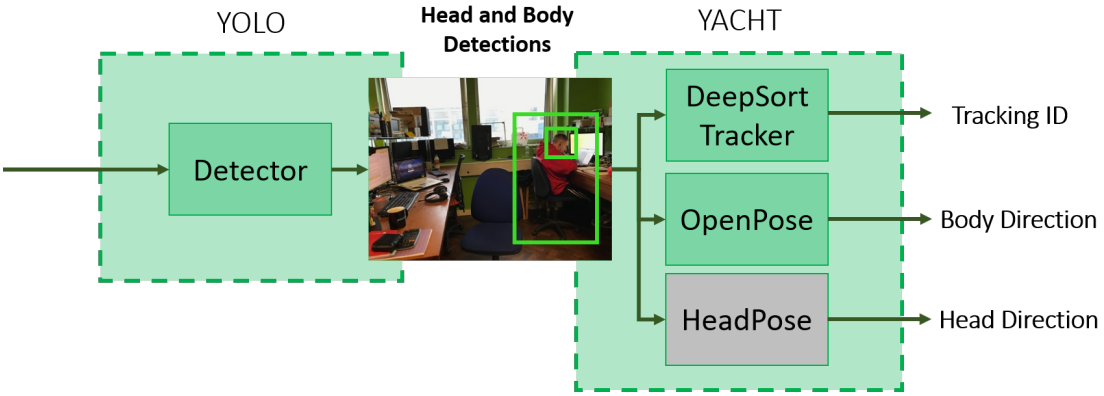
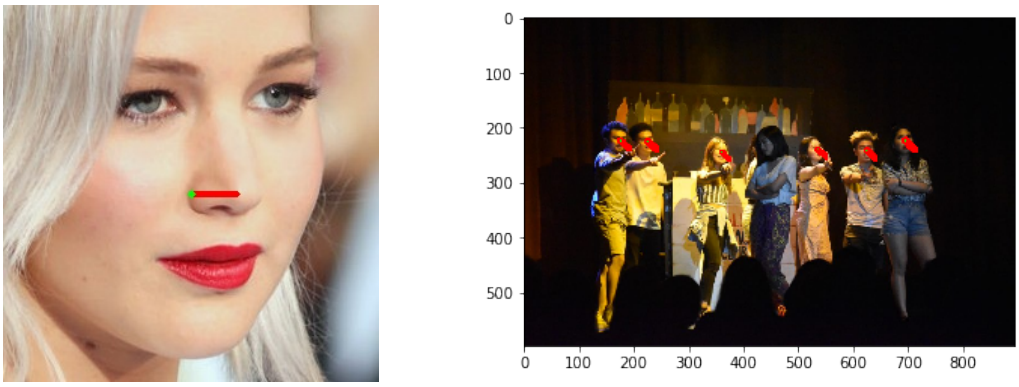


Figure 4.7: Initial HDD System with the HeadPose Node before removal.

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 estimation of the head pose. For smaller faces with multiple detections, the head pose estimation was not performed in real-time and also produced inaccurate estimations. 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 co-ordinates 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 co-ordinates of the objects, which it sends to ARTA for the reactive control of the wheelchair. This section covers how the system was designed so that it could 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. This allows Unity/UWP 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

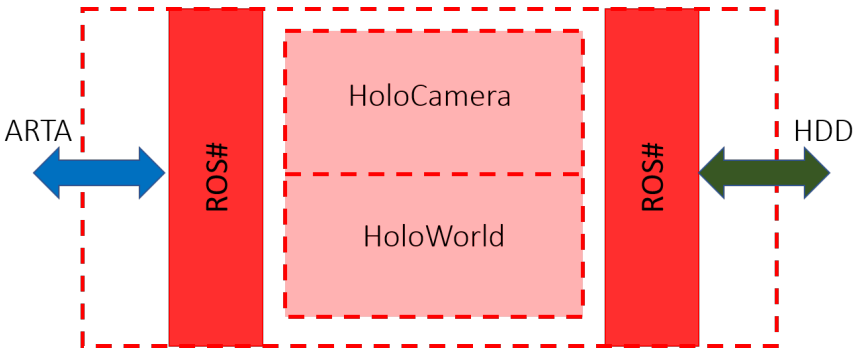


Figure 4.9: ROS# acts as a ROS wrapper around the Unity application, allowing for seamless communication with other ROS nodes.

the need for a video camera stream to be developed. 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, access 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 difficult 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 have 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 PWU in navigation by providing visual cues of people in the surroundings. Unity applications have the ability to place holograms in the users 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 co-ordinates 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 co-ordinates from the HDD system and projects the detected points into the world frame. This returns a set of world co-ordinates 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 robots frame. 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 has the ability to 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

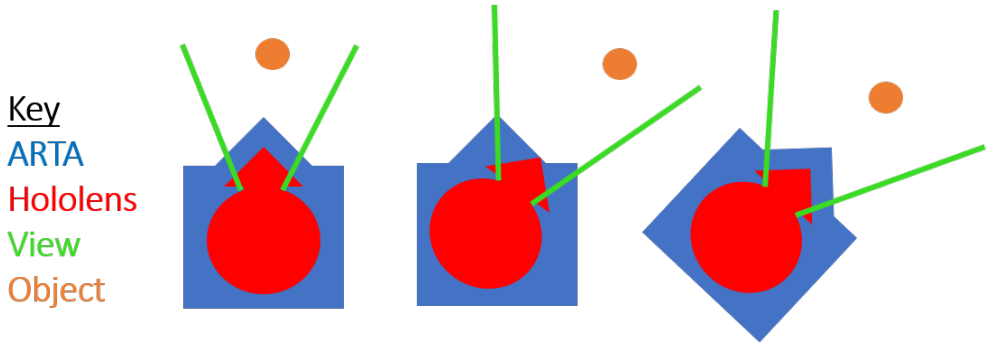


Figure 4.10: We need a way of knowing if a detected object is in front of the wheelchair, or if the PWU is looking to the side.

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.

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 [28, 29].

4.4.1 Breakdown

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 on 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.

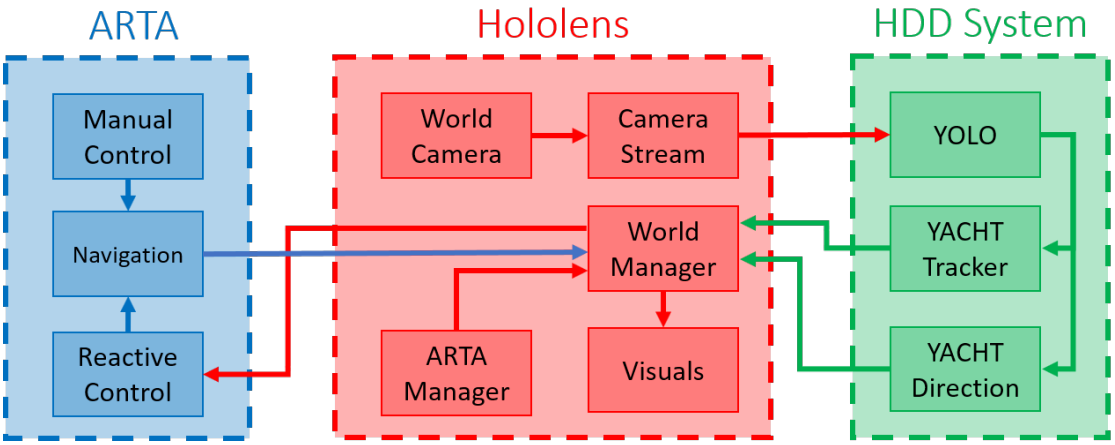


Figure 5.1: System diagram detailing individual components of the programs running on each device.

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.

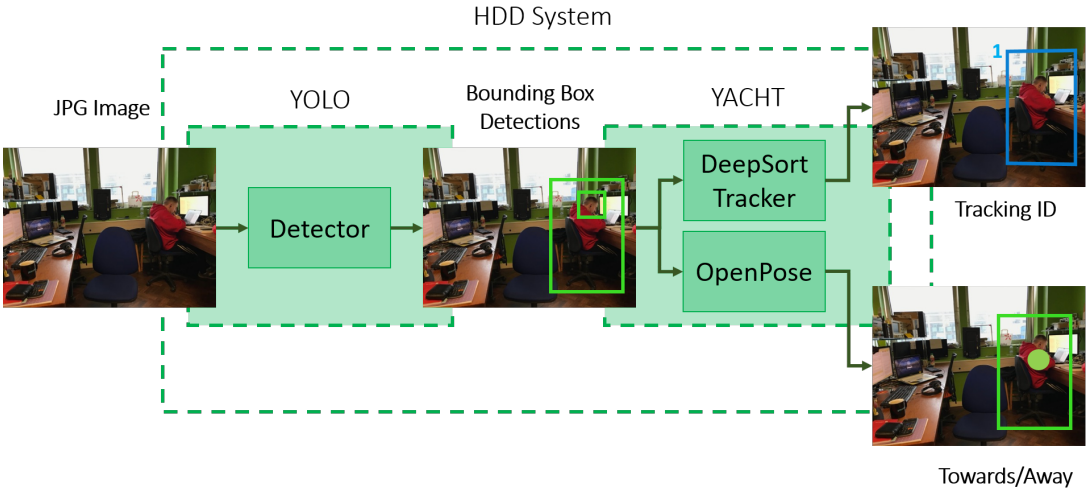


Figure 5.2: The HDD System is sub-divided into two ROS packages, YOLO and YACHT. We show that YACHT has two sub-nodes which both depend on the outputs of the detector.

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 key ones. A complete listing is available in the appendix. 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 [40]. The source code for the framework is freely available on GitHub, and it can be used to train different neural network architectures in a manner similar to 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 an $S \times S$ 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 5.20. 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 certain 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 [36] is a benchmark dataset to better evaluate detectors in crowd scenarios. The most important features of the dataset are the

size, quality of the annotations and the 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 [36], 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 the purposes of this training, we chose to use the full body and head boxes only. We wanted the detector 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 2 classes, head and body, as well as to use batches of 32 divided into 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 in order 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 in order 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 around 29.134. We reasoned that the network had reached a minima, 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 the 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 captured several more videos outside the EEE building and inside the 5th floor ICRS lab.

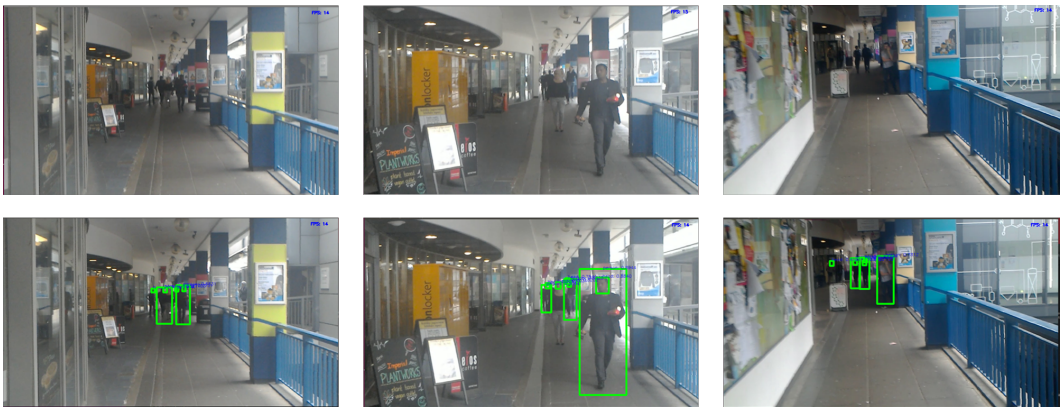


Figure 5.6: Evaluating the model on Sherfield Walkway test video. We notice that it is able to small figures close together at a distance.

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 further 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 attempt 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 seperate topics and comparing timestamps on individiual messages.

5.1.4 YACHT: Tracker

As explained in Section 5.1.3, the YACHT tracker depends on the Deep SORT algorithm [12]. 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.

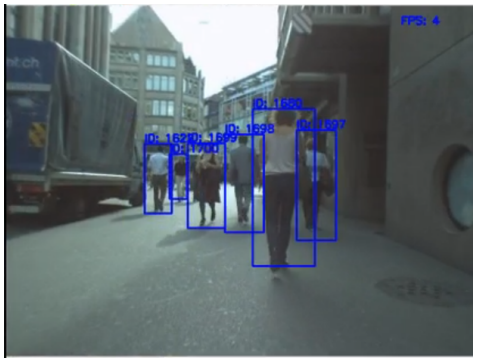
Deep SORT

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 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 so as 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.

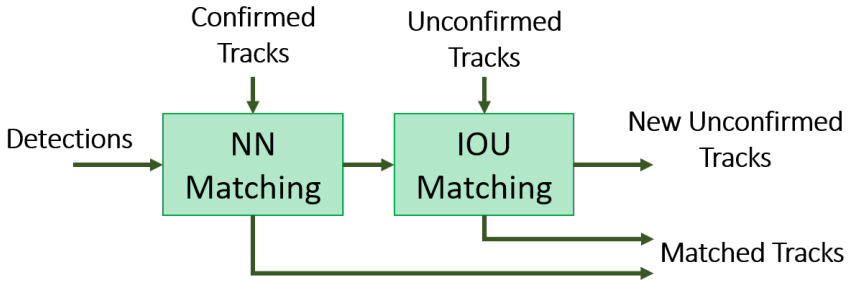


Figure 5.8: Visualization of Deep SORT matching. IOU matching is used as a final check for tracks that are unmatched by feature vector Nearest Neighbour matching.

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 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 [36]. 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 [41]. The library is well documented, and provides its own instructions on how to setup 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.

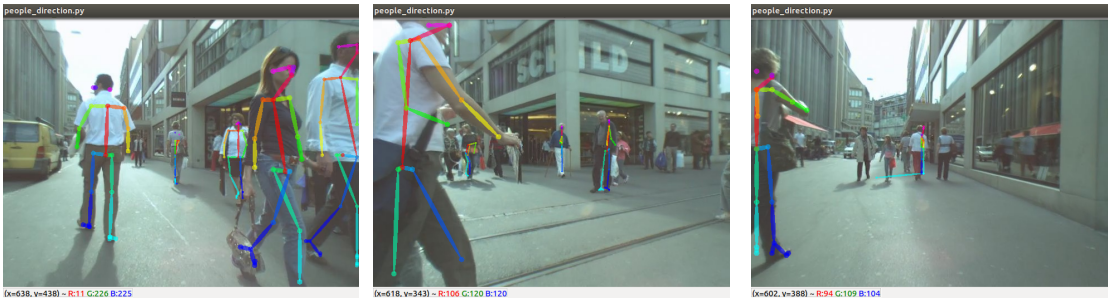
KeyPoint Estimation

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 on people further away is due to the network resolution being lowered. 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.

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



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

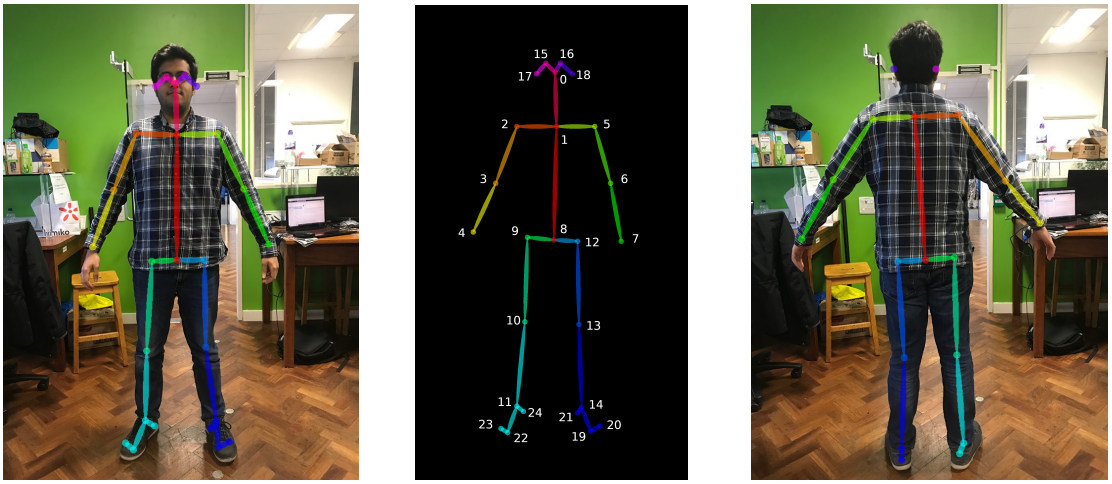
(b) On people at a distance, the estimation accuracy is limited by the network resolution.

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

Figure 5.9: Comparison of keypoint estimation at different scales

Defining Direction

By comparing the relative positions of certain key-points, we can determine if a person is facing towards 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.



(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 is able to 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.

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

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.

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 these are assigned track IDs by the tracker node. 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.

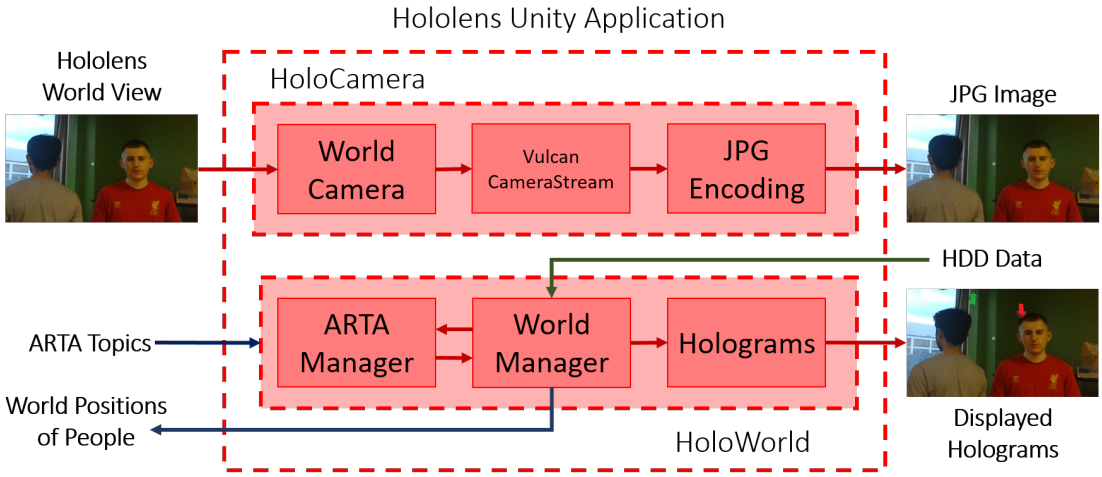


Figure 5.11: Components of the Unity application running on the Hololens. We divide the program into two sub-programs, HoloCamera and HoloWorld.

Due to the importance, we begin this section by describing the implementation of the video camera stream. We explain the libraries used, the the workarounds required to produce the stream and the limitations of using a Unity based approach. Afterwards, we explain the Unity application responsible for producing holograms, positioning 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 has the ability to 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 main programming language for the engine is C#, and the engine features built-in Windows Mixed Reality and Hololens support, allowing for easy 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

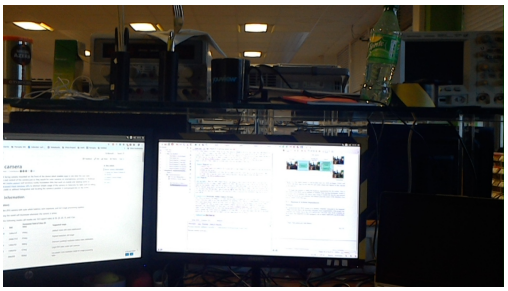
The images captured by the front facing camera is meant to be a representation of what the user is seeing. In reality, the field of view (FOV) of the front facing camera is greater than the actual FOV of the device, meaning the camera can see more of the world than the user can.

Specifications

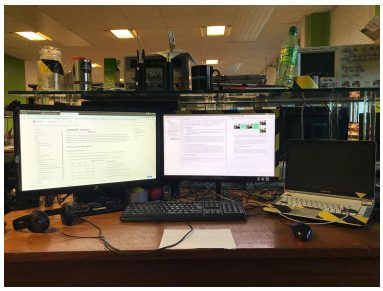
The front facing camera is a 2 mega pixel photo/ HD video camera with auto white balance, auto exposure and a full image processing pipeline. Due to the real-time requirement of the project, we chose to use the smallest video resolution possible in order to achieve the fastest frame rates. This limits 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 actually limits what the user can see in the surrounding area.

5.2.3 Hololens Video Camera Stream

For this project to begin, it was absolutely 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 developed 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 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.

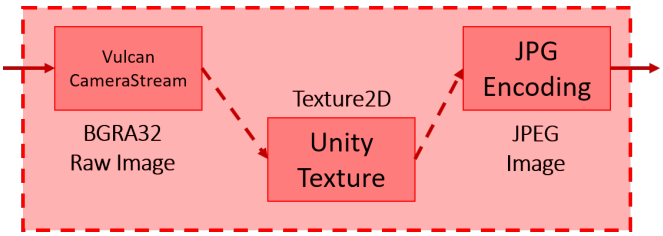


Figure 5.14: The image capture pipeline involves an intermediate Texture2D Unity state.

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

The captured frames are in the Microsoft standard `BGRA32` pixel format, which has 32 bits per pixel. The 4 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 has the ability to 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 so as to not 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.

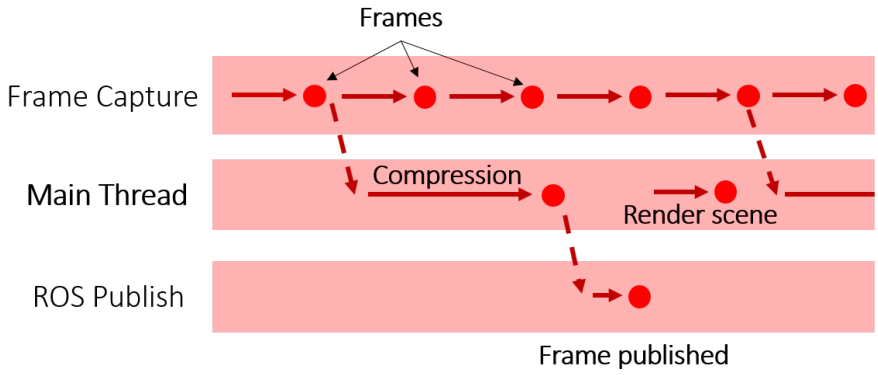


Figure 5.15: Image compression is a time consuming operation that must be done in the main thread. Only after compression is done can the holograms be rendered, limiting the application to 5 FPS.

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 PhD 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 partially 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

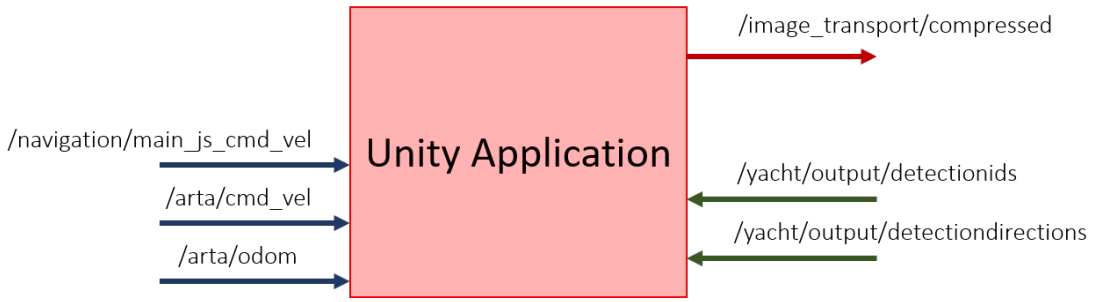


Figure 5.16: ROS topics in and out of the Unity application. Each topic carries data used by different parts of the overall system.

Compressed Images The raw video frames captured by the front facing camera are compressed into JPEG frames that can be published in the form of `CompressedImage` messages across ROS topics. Within the Unity application, we have setup `CompressedImage` publishers which are called when an 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 have the ability to 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' internal 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 their own 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 co-ordinates of the detections, as well as a tracking ID and whether the detected person is facing the camera.

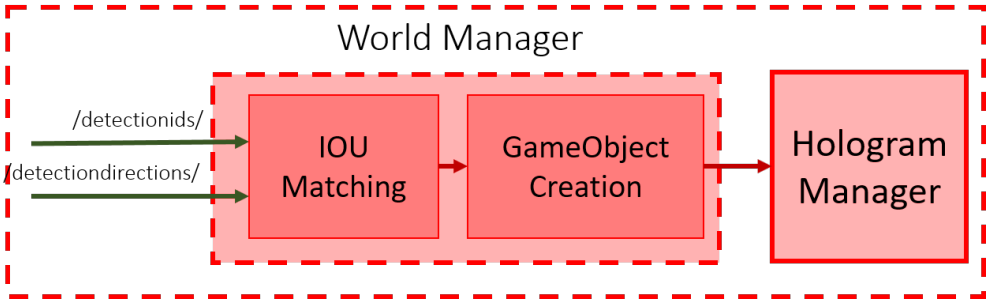


Figure 5.17: The Intersection-over-Union metric is used to compare bounding boxes for matching. The higher the metric, the more overlap between the boxes.

Matching Bounding Boxes Due to the design of the YACHT package, two different nodes determine the tracking ID and direction of a detection. This was done to allow 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 exactly since one message may be compared with another message from a 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 co-ordinate frame transforms to convert the point in the Camera co-ordinate system to the World co-ordinate system. We highlight the process in Figure 5.18.

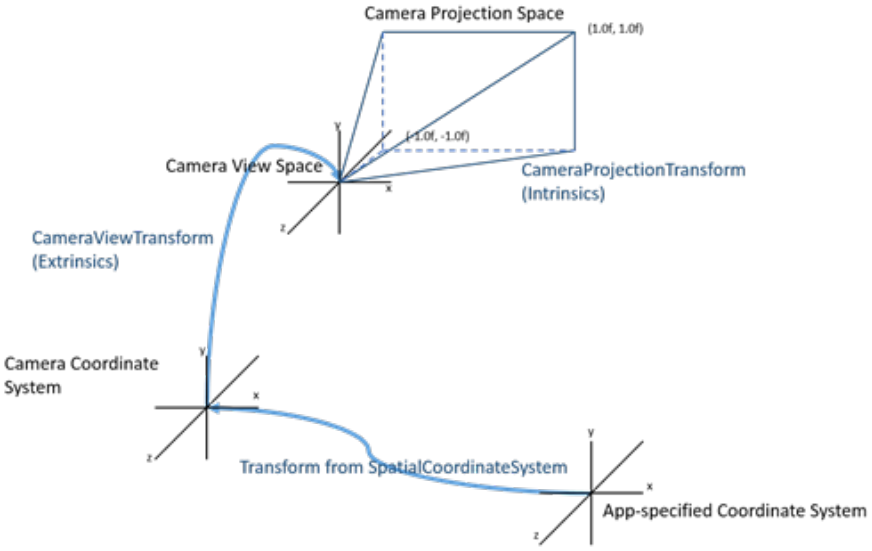


Figure 5.18: The front facing camera projects the surroundings onto a 2D plane. We can determine the real world position of a pixel by reversing the projection and using frame transforms.

Fortunately, the Vulcan Technologies library used to capture the frames has a function that converts pixel locations to world space co-ordinates. 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.

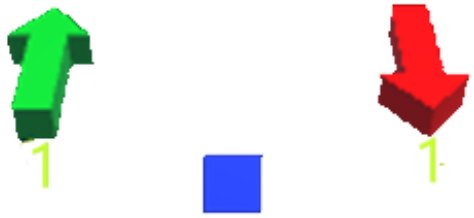
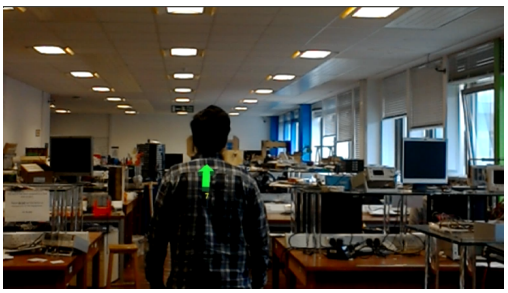
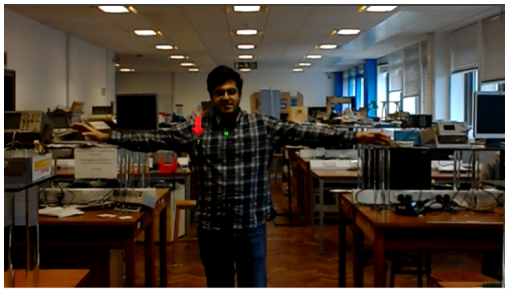


Figure 5.19: The Green and Red Arrow holograms rendered. The number under the arrow is the associated tracker ID of the object. This allows us to track objects in the surroundings.



(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.

Bibliography

[1] Ya Li Hou and Grantham K.H. Pang. Human detection in crowded scenes. *Proceedings - International Conference on Image Processing, ICIP*, (September 2010):721–724, 2010.

[2] Dongdong Zeng, Ming Zhu, Tongxue Zhou, Fang Xu, and Hang Yang. Robust Background Subtraction via the Local Similarity Statistical Descriptor. *Applied Sciences*, 7(10):989, 2017.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. Technical report.

[4] Massimo Piccardi. Background subtraction techniques: a review*. 2004.

[5] Manato Hirabayashi, Shinpei Kato, Masato Edahiro, Kazuya Takeda, Taiki Kawano, and Seiichi Mita. GPU Implementations of Object Detection using HOG Features and Deformable Models. Technical report.

[6] Paul Viola and Michael Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. Technical report, 2001.

[7] Navneet Dalal, Bill Triggs, Navneet Dalal, and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 886–893, 2005.

[8] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[9] Caglayan Dicle, Octavia I Camps, and Mario Sznaier. The way they move: Tracking multiple targets with similar appearance. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2304–2311, 2013.

[10] 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.

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

[12] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *Proceedings - International Conference on Image Processing, ICIP*, volume 2017-Septe, pages 3645–3649, 2018.

[13] 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.

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

[15] 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.

[16] 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-Janua, pages 3711–3719, jan 2017.

[17] 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.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 770–778, 2016.

[19] Leonid Pishchulin, Eldar Insafutdinov, Siyu Tang, Bjoern Andres, Mykhaylo Andriluka, Peter Gehler, and Bernt Schiele. DeepCut: Joint subset partition and labeling for multi person pose estimation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 2016-Decem, pages 4929–4937, 2016.

[20] Zhe Cao, Tomas Simon, Shih En Wei, and Yaser Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. In *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, volume 2017-Janua, pages 1302–1310, 2017.

[21] T Bailey and H Durrant-Whyte. Simultaneous localisation and mapping (SLAM): Part I - The essential algorithms. *IEEE Robotics and Automation Magazine*, 13(2):99–108, 2006.

[22] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[23] 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, 2017.

[24] David Nistér. A Minimal Solution to the Generalised 3-Point Pose Problem. *Journal of Mathematical Imaging and Vision*, 27(1):560–567, 2004.

[25] Paul Milgram and Fumio Kishino. A TAXONOMY OF MIXED REALITY VISUAL DISPLAYS. *IEICE Transactions on Information Systems*, E77(12):1–15, 1994.

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

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

[28] Mark Zolotas, Joshua Elsdon, and Yiannis Demiris. Head-Mounted Augmented Reality for Explainable Robotic Wheelchair Assistance. 2018.

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

[30] Sheng Jin, Xujie Ma, Zhipeng Han, Yue Wu, Wei Yang, Wentao Liu, Chen Qian, and Wanli Ouyang. Towards Multi-Person Pose Tracking : Bottom-up and Top-down Methods. *Proceedings of the IEEE International Conference on Computer Vision*, (2):4–7, 2017.

[31] Eldar Insafutdinov, Leonid Pishchulin, Bjoern Andres, Mykhaylo Andriluka, and Bernt Schiele. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9910 LNCS, pages 34–50, 2016.

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

[33] 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.

[34] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. Technical report, 2018.

[35] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft COCO: Common objects in context. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8693 LNCS, pages 740–755, 2014.

[36] Shuai Shao, Zijian Zhao, Boxun Li, Tete Xiao, Gang Yu, Xiangyu Zhang, and Jian Sun. CrowdHuman: A Benchmark for Detecting Human in a Crowd. 2018.

[37] Anton Milan, Laura Leal-Taixe, Ian Reid, Stefan Roth, and Konrad Schindler. MOT16: A Benchmark for Multi-Object Tracking. 2016.

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

[39] Massimiliano Patacchiola and Angelo Cangelosi. Head pose estimation in the wild using Convolutional Neural Networks and adaptive gradient methods. *Pattern Recognition*, 71:132–143, 2017.

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

[41] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional Architecture for Fast Feature Embedding. 2014.