University of Glasgow | School of Computing Science

Honours Individual Project Dissertation

Training the Baxter Research Robot using a VR interface based on the HTC Vive

Sean A. Skilling

March 26, 2019

Abstract

Every abstract follows a similar pattern. Motivate; set aims; describe work; explain results.

“XYZ is bad. This project investigated ABC to determine if it was better. ABC used XXX and YYY to implement ZZZ. This is particularly interesting as XXX and YYY have never been used together. It was found that ABC was 20% better than XYZ, though it caused rabies in half of subjects.”

Education Use Consent

Acknowledgements

I would like to thank my supervisor Dr Paul Siebert for his support and patience throughout this project. Also special thanks to my mum Yvonne, dad Paul and brother Callum - all of whom helped with proofreading.

Contents

1 Introduction

The goal of this project was to create a virtual reality interface to control the Baxter research robot that achieves telepresence using low cost consumer hardware and graphics software. Virtual reality systems have become more powerful and more accessible than ever before, which makes incorporating them into projects easier and more affordable.

The motivation behind this project is a desire create a fully immersive interface to connect with the Baxter robot. This would utilise a virtual reality headset and hand controllers to achieve telepresence, but would also have alternate uses such as training Baxter by example. Such a system could also be utilised for demonstrative purposes at the university. The system should control all input of movement through a single virtual reality system – in this case, the HTC Vive was chosen. In addition, the whole setup should ideally run on a single computer system.

The following chapter **Background** will explore relevent areas of existing research for insight in teleoperation using virtual reality interfaces and provide justification for the approach taken to achieve the goal of this project. Chapter 3 **Technologies** will then explore the various hardware and software components used in the course of the study. An overview of the system's major design decisions and implementations is provided in Chapter 4 **Design and Implementation**. Chapter 5 **Evaluation** will provide analysis of the system in terms of accuracy and performance. Finally, Chapter 6 **Conclusion** will discuss this study's achievements as well as areas in the implemented system which could be developed with further work.

2 Background and Technologies

The main aim of this project is to take the existing implementation of an immersive virtual reality interface for the the teleoperation of the Baxter robot and simplify it. In order to justify the approach taken in the context of the study, analysis of relevant work within available literature and conducted by the same institution is required. In particular, relevant work in Spatial Presence in VR, Systems for Tracking Human Movement and Human-Robot Mapping will be presented. This chapter will also detail the hardware and software technologies that were used in the project as well as any issues that we envision may cause problems for this system upon implementation.

2.1 Background

2.1.1 Spatial Presence in VR

Due to the relative newness of the commercial virtual reality field, the majority of research into immersion in virtual reality presents it in context of video games, such as Bracken and Skalski (2010). Despite this specification, a great deal of insight can be drawn from these studies in regards to immersive techniques. In particular, these studies are the origin of the term "Spatial Presence" which describes the feeling of physically existing in a virtual space (Weibel and Wissmath (2011)). In some teleoperation contexts, this is often referred to as "telepresence".

This labelling of immersion as "spatial presence" or "telepresence" could be argued as being misleading. While telepresence has been identified as a fundamental component in driving immersion (Siebert and Shafer (2017)), there are several other factors that need to be taken into account such as the accuracy of movement tracking and the latency in the system. Failing to account for these key factors could ruin an otherwise perfectly realistic virtual world and break the immersion for the user.

It is also wise to compare how immersive virtual reality setups are in comparison to more standard setups. In research by Siebert and Shafer (2017), they compare a VR HMD with traditional monitors for creating immersion for the user in video games. The results show a significant increase in spatial presence when using a virtual reality system compared to standard monitor displays. However, virtual reality systems do have their own drawbacks. Specifically, virtual reality interfaces have been known to cause some users to experience a feeling similar to motion sickness, dubbed "cybersickness" (Vinson et al. (2012)). While the focus of this project is not on cybersickness, it should be noted that it was cited as an obstacle for VR adoption and for the application of this technology in certain aspects.

2.1.2 Systems for Tracking Human Movement

In order to obtain the data required to map the user's movements to the robot, it is necessary to track the user's movements. There are multiple different ways of tracking and record movement, but the two covered here are full-body tracking and room-scale tracking. Full-body tracking

devices, such as the Kinect produced by Microsoft, use depth cameras to create a real-time interface for teleoperation. The Kinect in particular has often been praised in studies for its tracking abilities without relying on complex sensors, achieving this through recognizing the joints and poses of the user captured from depth images. Figure 2.1 shows the joints which are tracked in this process. This isn't enough for the Kinect to be able to grant full telepresence on its own since it lacks a display, although some studies address this limitation by pairing it with a VR headset such as the Oculus Rift (Hsu et al. (2013)). The Kinect also has issues with tracking; Other studies, such as the study by Xu et al. (2016), encounter the same display shortcoming when using the Myo armbands and also require an additional VR headset to achieve telepresence. The skeletal tracking of the Kinect also has several flaws, such as the sun's rays interfering with the infrared sensors and an inability to accurately track users who are either not facing the Kinect or too far away.

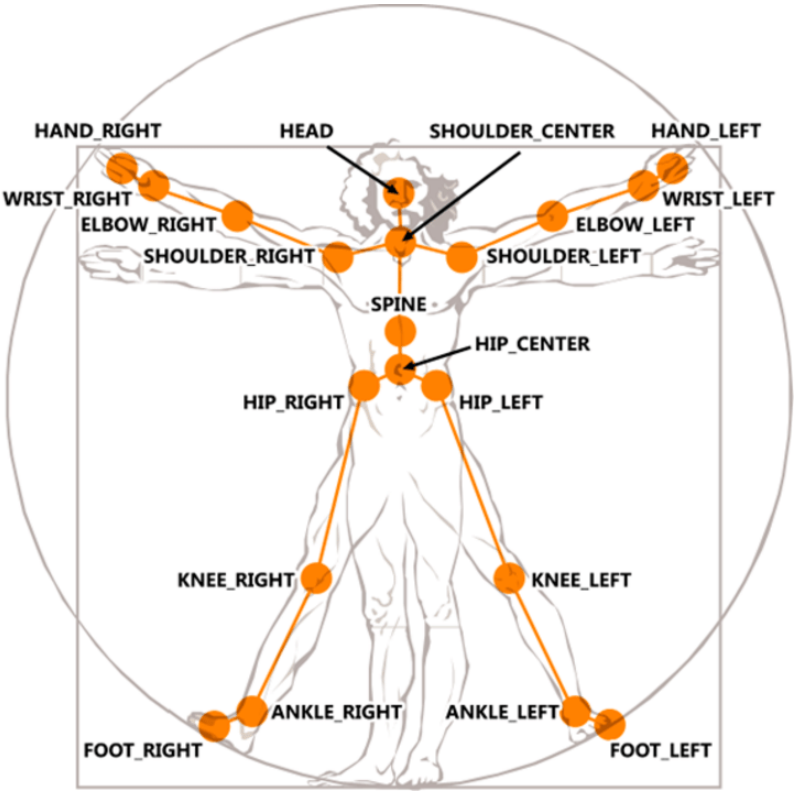

Figure 2.1: *The joints tracked by the Kinect's skeletal tracking system. Taken from Hsu et al. (2013)*

Virtual reality systems such as the HTC Vive, on the other hand, utilises room-scale tracking through the use of stationary sensors which, along with sensors built in to the various components of the system (discussed in more detail in Section 3.1) create a 3D co-ordinate system that allows the system to represent the user's orientation and position within a "play area" with diagonals of 5 metres. The HTC Vive system also already contains a display in the form of the Vive HMD, negating the reliance on an additional system for the display that the full-body systems discussed all had. The Oculus Rift, the Vive's main competitor, is another VR system which can be similarly be used for teleoperation purposes. Such is the case with a study conducted at the University of Glasgow by Vanik (2017), which justified the system's usability for tracking and mapping to the Baxter robot. However, the study fails to provide details for the system's accuracy.

A study by Niehorster et al. (2017) was conducted analyse the HTC Vive and its capabilities in

tracking human movements. It found that the Vive combines efficient and accurate tracking with low latency, but also that there were several shortcomings with the system. Notably, the Vive tracks its virtual space by the "ground plane" which has its distance from the headset calibrated during Room Setup and is always taken from the base of the headset which creates innaccuracies when reading the roll and pitch data. While this can be fixed by hardcoding in an offset, this is lost whenever the tracking has to be re-calibrated. However, this is a result of moving whilst operating the system; seated and standing users can mitigate this problem to an extent.

2.1.3 Human-Robot Mapping

There are multiple models of mapping the movements of a human user to a robot. In the context of this project there are two particular models which bear further analysis: robocentric and egocentric. The robocentric model has the user co-existing in a virtual environment along with the robot as they control it, with the robot and the user existing as separate entities in this space. However, the egocentric model describes the system where the user and robot exist as a single entity in the virtual space - the user essentially "becomes" the robot when interfacing with it. This model minimises the coupling between the user and robot in an attempt to mitigate the discomfort and limitations caused by the robocentric model of control.

In a study by Lipton et al. (2018), which compared an egocentric approach they dubbed the "homunculus model" against several other models, the mimicking model was found to have the highest degree of immersion due the virtual space superimpoding the user's state and movements onto the robot. This is partly why previous projects in this field by this institution used the mimicking model such as the study by Vanik (2017). This model works best on robots with a humanoid form such as Baxter. This is due to the similarity in shape and resulting ease in mapping movements, being mostly direct with some differences due to differing numbers of joints in the robot's limbs.

2.2 Technologies

2.2.1 Baxter

Baxter is an industrial robot, built and initially released by Rethink Robotics in September 2011. Mimicking the appearance of a human torso, arms and head atop a stand, Baxter's primary purpose is in automating industrial sized processes at an affordable cost. It comes with its own Software Development Kit (SDK) containing the various tools for interfacing with the different components of the robot. In the context of this project the most relevant of these packages is the Inverse Kinematics (IK) Service, which calculates a path of movement for its arm from its current location to the desired point given just that end point.

Figure 2.2: *The Baxter robot.*

For the purpose of mapping from human users to robots, Baxter isn't an perfect candidate due to a different number of joints and degrees of freedom to the user – human arms only have six degrees of freedom due to their joint structure, while Baxter's arms have seven degrees of freedom. These joints (show in Figure 2.3) are also positioned differently than those in a human, allowing it to reach poses that a human is incapable of reaching.

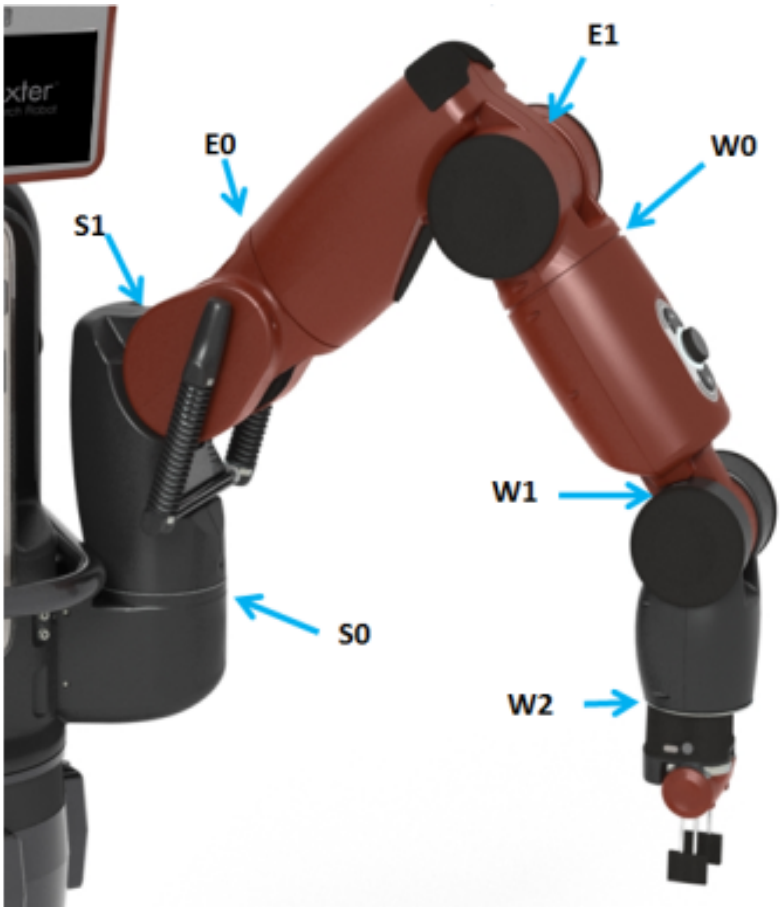

Figure 2.3: *The seven joints in Baxter's arm. The joints labelled S are the shoulder joints, E joints are the elbow joints and W joints are the wrist joints. Notably, Baxter has an extra degree of freedom in its wrist compared to a human*

This causes some issue with the mapping of human movements to Baxter. Using the IK Services, the movements are executed by calculating the path to the endpoint which could be inconsistent between the user and the robot. This may lead to instances where the point mapped doesn't match and fails to mimic movements exactly. There is also the consideration that Baxter wasn't designed for human-robot mapping, instead having industry tasks in mind.

Despite the above potentially causing issues during the project, Baxter was chosen for this project due to his affordability compared other similar robots (costing Âč19,000 compared to other industrial robots costing upwards of Âč37,000) and his design being close enough to shape of a human for accurate mapping. In addition, the Baxter robot was already available, saving a large sum of money purchasing a robot. Baxter also has a Gazebo simulation (a virtual environment containing a functioning replica of Baxter) which removes the need to have the Baxter robot on hand for development and allows work to be done away from his location.

2.2.2 ROS

ROS (**R**obotic **O**perating **S**ystem) is best described as robotics middleware with its main support base being for Ubuntu Linux systems. It contains code packages for various programming

languages which allow many different varieties of robots to be driven and controlled, including Baxter. Since ROS (and by extension Baxter) is restricted mostly to Linux, this is the major limiting factor in our options for the other components of the project (there do exist "Experimental" packages for Windows, MacOS and Fedora Linux systems, but that is outside of the scope of this project).

A major component of working with ROS is the use of subscriber nodes and publisher nodes. Publisher nodes collect data from various functions and "broadcast" them into the system, while subscriber nodes are constantly listening for their specific function calls in these broadcasts. When the published nodes broadcast is picked up by the subscriber node, the data is then handled by any functions pertaining to them by the subscriber node before handing off that data to the target (Baxter in this case).

ROS subscribers and publishers can be written in a variety of programming languages, with the three main client libraries being C++, Python and Lisp. Python was chosen for this project due to its ease of readability and faster development due to simpler code. In addition, the developer has experience in Python but no experience in C++ or Lisp making the decision a straightforward one.

2.2.3 HTC Vive

The HTC Vive is a virtual reality headset system developed by smartphone developer HTC in partnership with Valve Corporation, a long-standing fixture in the videogame industry responsible for both highly popular games and one of, if not the most used digital storefronts in Steam. The system consists of three components which work in tandem to achieve accurate room-scale tracking of a user's movement in 3D space as seen in Figure 2.4: the HMD (**H**ead-**M**ounted **D**isplay), the controllers and the base stations. By having the HMD and controllers in view of both base stations, the systems is able to track the users movements with relatively high accuracy using infrared detectors and emitters respectively.

Figure 2.4: *The major components of the HTC Vive: the HMD (center), the controllers (front-sides) and the base stations (back-sides).*

The main reason the HTC Vive was chosen over other virtual reality systems such as the Oculus Rift used by Peter Vanik in his project is due to its compatibility with Linux systems. The Rift is only supported on Windows at this time and would require an additional system running on Windows alongside the system to run ROS (which is only supported on Linux systems), which goes against the point of the project. The Vive was updated to support Linux in early 2017 [Source], which makes it the ideal choice since one of the aims of this project is to limit the

number of required systems to just a single Linux machine. Even so, the post-launch support for the Vive on Linux has been relatively poor - two years after launch performance is still worse on Linux than it is on a system with identical specifications that runs on Windows, and with only a handful of supported games and other software.

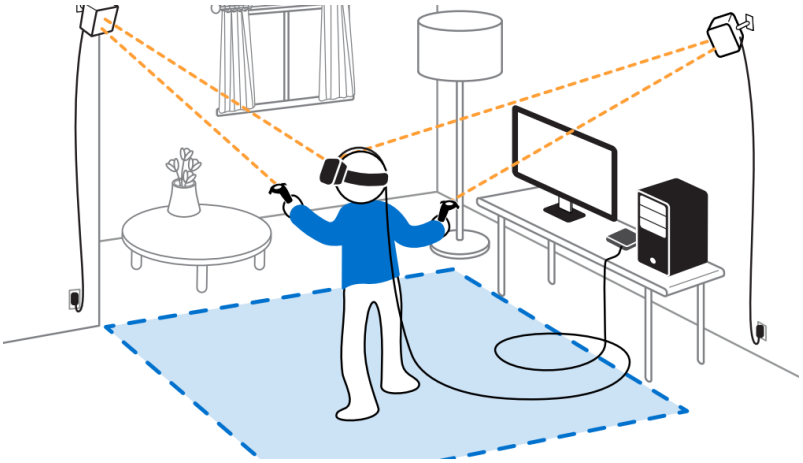

Figure 2.5: *The diagram above shows the intended layout of a room when using the HTC Vive, with both base stations raised into opposing corners of the room. The system is intended for use in spaces with 5 meter diagonal.*

The HTC Vive has several other limitations, some of which are more glaring in the context of this project than others. The base model of the Vive is limited in its range of motion by its dependence on the HDMI and USB cables connecting them to the computer unless the hefty cost for a wireless adaptor is deemed worthwhile. In addition, the HMD itself is fairly heavy at roughly 0.5 kilograms and could cause strain or injury over long user sessions. The tracking system also proved surprisingly easy to disturb, as any non-Vive system item that sits within the space between the controllers, the HMD and the base stations would cause the system to lose track of the various components and either desync them or lag heavily. The system is also rather inconvenient to set up as intended, as it requires drilling into the walls to attach the included wall mounts as shown in 2.5 although this can be done by attaching the mounts to microphone or lighting stands for a less permanent fixture.

2.2.4 SteamVR

Launched simultaneously with the HTC Vive, SteamVR is a virtual reality service created and distributed by Valve to interface with the Vive as well as other VR systems utilising their OpenVR API.

It would be an understatement to describe SteamVR on Linux as being unstable. Most of the defined issues were encountered when trying to set up the HTC Vive, aside from a few Windows-specific errors and crashes, as well as several error codes that were not defined anywhere in the support for SteamVR. Another major issue was that, due to virtual reality on Linux still being in its infancy, most of the fixes and solutions for the issues were either only explained through Windows or didn't work on Linux at all. This led to repeated testing sessions cut short by having to uninstall Steam and SteamVR and reinstalling both, which could take upwards of two hours each time when factoring in various issues with installing Steam itself. In addition to this, a conversion between OpenVR's co-ordinate system (shown in Figure 2.6) and the HTC Vive's co-ordinate system is required due to their differences.

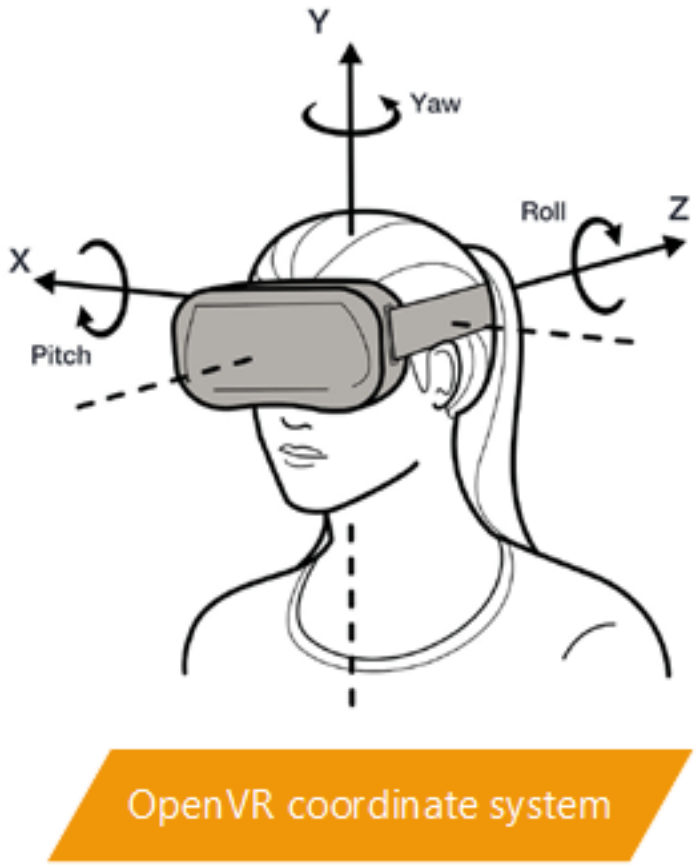

Figure 2.6: *The figure above shows the OpenVR co-ordinate system in relation to the HTC Vive's HMD. Notably, the system reads moving forwards as a negative movement in the Z-axis, with the X-axis relating to horizontal movements.*

Despite the issues mentioned above, SteamVR was chosen as it was the earliest method of creating virtual reality on Linux and it was hoped that it would be more stable than the alternative of Proton, which allows the Windows version of Steam games to be run on Linux through a combination of Wine and Windows partitions.

2.3 Summary of Background and Technologies

Virtual reality systems, though primarily designed for playing video games, are beginning to be introduced into robotics projects and while the findings aren't all applicable to robotics several results can be helpful to the field. In particular, the research into immersion and cybersickness can be used to better understand these aspects of the field, though this is mostly outside the focus of this project. In the particular case of cybersickness, the benefits of VR vastly outweigh the drawbacks and as such we accept it as a risk.

When using virtual reality systems to track user's movements, devices with integrated displays such as the HTC Vive and Oculus Rift are far more useful due to not needing to employ a second device with a screen even when taking into account full-body tracking devices compared to

room-scale tracking. In addition, since the Baxter robot used in the project consists of just a torso and head and is incapable of movement, full-body tracking is excessive for this project and the room-scale devices are acceptable.

Both the robocentric and egocentric models of teleoperation have their advantages as seen in the project by Lipton et al. (2018). However, due to the focus on immersion when the user is interacting with the robot, the egocentric model is more appropriate as it allows the user to "see" out of the robot's eyes (through the use of the integrated HMD) and become the robot. In addition to this, the mimicking model will also be applied when dealing with the mapping from user to robot because of its increased speed of development and ability to achieve telepresence in contrast to the homunculus model. This is in addition to the latter system having far more complexity, further justifying the mimicking model.

Due to the integration of Baxter being a key point of the project, choosing a suitable robot wasn't a factor. However, the use of Baxter creates a separate issue as it limits our choices for the other technologies available. ROS is only officially supported on Ubuntu Linux systems, which restricts our option of virtual reality systems. There are two primary options for VR on Linux: using the HTC Vive through SteamVR or through the use of Proton. Due to the pre-existing access to a HTC Vive system and an uncertainty as to whether or not Proton would support the project, it was decided to use the HTC Vive and SteamVR. The following chapter will discuss how the technologies influenced and shaped the design and eventual implementation of the teleoperation system.

3 Design and Implementation

With the technologies covered in the previous chapter, this chapter will cover the iterations of design of the project and their implementation from the first ideas and experiments to the final product, as well as the various issues encountered along the way.

3.1 Requirements

In this section, we list the functional requirements using the MoSCoW method.

3.1.1 Must Have

- Implementation of a mapping from the HTC Vive to the Baxter research robot

3.1.2 Should Have

- A visual feed system that works on the Baxter Gazebo simulator.

3.1.3 Won't Have

- A working implementation of a visual feed from the Baxter research robot and/or the ZED camera to the HTC Vive

3.2 Design

In its simplest form, the teleoperation consists of four components:

- **User Controller** - the virtual reality system which will track the movements of the user. This is also where the processed visual data is used. In this system, this is fulfilled by the HTC Vive system.
- **Movement Processing** - the section which takes the movement related data, such as co-ordinates and rotations, and works with them until they can be passed on
- **Vision Processing** - the section which takes the vision related data, which consists of the video feed from the camera attached to Baxter, and works with it until it can be passed on
- **Robot** - the robot to which the movement data is mapped. This is also the source of the video feed. In this system, this is fulfilled by the Baxter robot.

This simplified architecture is shown in Figure 3.1. While this model is oversimplified, it also identifies the key areas for work within the system and divides the interface into two clean halves – one focused on tracking the movements and mapping them to the robot and the other on recording the visual feed and transmitting it to the Vive.

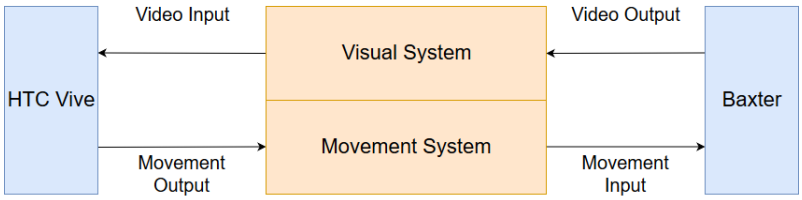

Figure 3.1: *The proposed system in its earliest and simplest iteration. As of writing the HTC Vive is the only Linux-compatible VR system – hence why it is specified in the diagram. A two-computer solution could replace the HTC Vive with a different headset.*

From this model, it becomes easier to envision the system with the hardware and software detailed in Chapter 3. There need to be three different applications within the system to handle the major hardware elements – the HTC Vive and Baxter. These are:

- **HTC Publisher Application** – OpenVR and SteamVR interface to retrieve data from the Vive devices, which are then published as ROS messages (TFMessage, Joy and Posestamped) on to different ROS topics
- **Baxter Subscriber Application** – various ROS topics are used to gather the data sent by the publisher and processed to obtain the goal pose for the robot, which is used by the IK service as a parameter. Each subscriber acts as a separate thread within the application running different function calls depending on which section of the system it deals with.
- **Camera-HTC Application** – SteamVR interfaces with the HTC Vive to render the captured image streams from both lenses of the camera on the respective lens of the Vive HMD

Applying this information to the initial diagram helps create a clear architecture of the teleoperation system. This is shown in Figure 3.2, where the camera portion of the system is fulfilled by the ZED camera.

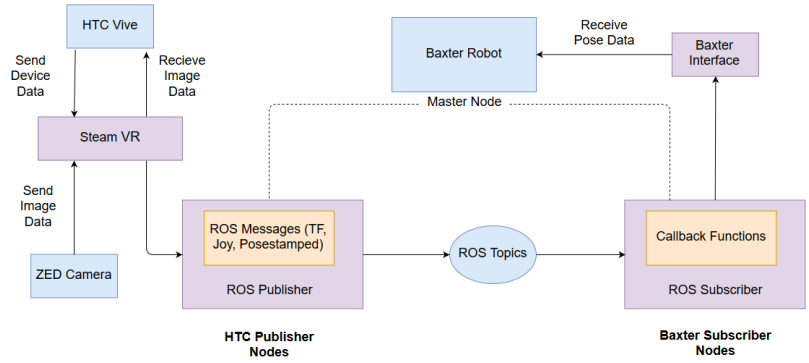

Figure 3.2: *The finalised system architecture. In the physical system the ZED Camera (or similar camera system) would be connected to the Baxter robot, however they were separated for the clarity of the diagram.*

While this design appears to be functional, there is an issue with working around SteamVR and its limitations. SteamVR is unable to interact with multiple applications at the same time, with

the newer application overriding the older one. This is presumably to prevent issues arising when attempting to play multiple games at the same time using the Vive and while it makes sense it does have an effect on the system. The video feed and movement data both have to be accessed by their respective applications through SteamVR, which means that both data streams need to be fed through a single file or interface. There are several ways around this issue which are discussed in Section 6.2, however due to time constraints the video feeds had to be put aside to ensure that the movement systems were completed. This means that while the teleoperation system does include a workaround for this constraint, it is highly flimsy and could be considered unsuccessful.

3.3 Implementation

3.3.1 HTC Publisher

The ROS package **htc_publisher** contains the software for handling the publishing of data from the HTC Vive to the relevant topics. This software is separated into two scripts: **tf_and_joy** and **frame_as_posestamped**. After initalising the ROS node for the application with the Master node operating Baxter using a Python script, TFMessages (the standard format ROS uses to handle transformations) are published containing the poses of the HTC Vive from every frame of reference it has at that time. A second Python script is then used to initialise additional ROS nodes which are used to allow the controller frames to be cross-referenced against another frame that will be used as the user's origin for calculation. This function is also where specific button presses are detected and published as Joy messages. Much like the HTC Vive itself, the application requires an instance of SteamVR to be running and uses a SteamVR runtime argument when starting. For convenience and to simplify the running of the application, a launch file - **htc_publisher.launch** has been included and comes with preset parameters.

The **tf_and_joy** script is the script responsible for creating the initial ROS node. In order to do this, it initialises the OpenVR API so that it can interface with the HTC Vive device through SteamVR. Using a loop, the script checks to see if the Vive system is connected four times before killing the process. If the Vive system is found, another loop then checks for the controllers to make sure that they are also connected. This loop does not kill the system if it cannot find the controllers, as there is always the chance a controller may simply be outside of the sensors range (for example, sitting on a desk while the application is being launched). When all the parts of the HTC Vive have successfully connected, the ROS node is initialised.

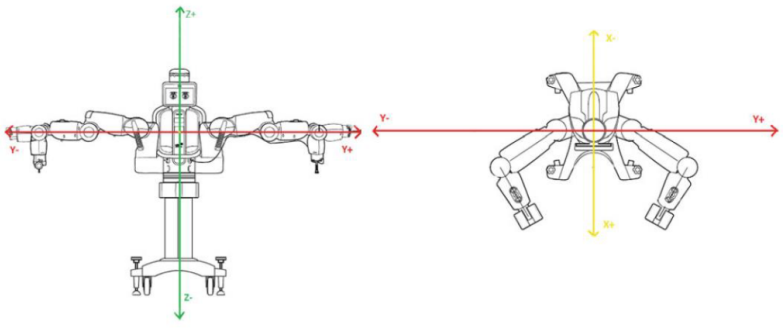

Figure 3.3: *A replication of the co-ordinate system used by Baxter and ROS. Taken from Vanik (2017).*

With the node initialised, a TransformBroadcaster is created to handle the transformation of the pose data. The OpenVR and ROS co-ordinate systems, shown in Figure 2.6 and Figure 3.3

respectively, do not align. To covert from the OpenVR system to the ROS system we employ the function in Listing 3.1 which makes use of the the transforms shown in equations 4.1 through 4.3. In order to do this, a 3x4 matrix is created and filled with floats representing the Cartesian co-ordinates and quarternion rotations for all of the connected devices through the use of an OpenVR function *mDeviceToAbsoluteTracking()* to extract the pose data from the controllers.

$$X_{ROS} = -Z_{VIVE} \tag{3.1}$$

$$Y_{ROS} = -X_{VIVE} \tag{3.2}$$

$$Z_{ROS} = +Y_{VIVE} \tag{3.3}$$

The button inputs are also handled by this script, which converts the inputs into Joy messages. The inputs are separated into whether it is a binary button (such as the menu button on the controller) that returns 1 if the button is pressed or with variable pressure (such as the trackpad) that returns a value in a range from 0.0 to 1.0. The inputs are also separated based on which controller the button is in. The triggers operate Baxter's grippers, while the trackpads activate "roll mode" in the arm where the arm's movement is disabled and rotation is enabled. With the HMD and controllers of the HTC Vive being the basis of the mapping for Baxter, the system allows control over Baxter which will be assessed in the Evaluation chapter.

The **frame_as_posestamped** script, on the other hand, takes each controller and publish them as PoseStamped messages - which uses the frame of one device as a reference of an other. This can be done at a rate of up to 250 times every second. In the context of this project, the script publishes the left and right controllers as PoseStamped messages that use the HMD as a reference frame and using the HMD as the user's origin.

3.3.2 Baxter Subscriber

The ROS package **baxter_subscriber** uses several Python scripts to retrieve the data from the HTC Vive and SteamVR that was published by the **htc_publisher**. Unlike the publisher application, the subscriber works around a central main application which initialise the Baxter robot (either the real machine or the simulated model) as well as the various subscribers to retrieve the data from the published topics. Either the Baxter robot or the Baxter Gazebo simulation is required for the subscriber to run, much like the publisher's reliance on SteamVR. Much of this package is derived from the default baxter_subscriber package provided with Baxter and as such only the more relevant scripts will be discussed in further detail.

The **subscriber.py** script handles the subscribers that collect the data from the published messages defined by the publisher application, shown in Listing 3.2. It contains functions that edit aspects of the data that weren't covered in the publisher before being published, such as correcting the rotational mapping of the controllers to Baxter. Baxter and ROS use a different orientation of rotation than the Vive and OpenVR, similar to their differing co-ordinate frames. As such, it is necessary to transform the rotations as detailed in the equations 4.4 through 4.6. These transformations are derived from the study by Vanik (2017), where he also noted that the Oculus Touch controller required an extra +90° rotation in Pitch due to a difference in mapping - Baxter's neutral end effector pose is at a different position relative to its "hands" than that of the controller. This also applies to the Vive's controllers. meaning the same transforms can be used.

$$Roll_{BAXTER} = -Yaw_{VIVE} \tag{3.4}$$

$$Pitch_{BAXTER} = -Pitch_{VIVE} + 90° \tag{3.5}$$

$$Yaw_{BAXTER} = -Roll_{VIVE} \tag{3.6}$$

```
def from_matrix_to_transform(matrix, stamp, frame_id, child_frame_id,
                     to_ros_reference_frame=True):
   t = TransformStamped()
   t.header.stamp = stamp
   t.header.frame_id = frame_id
   t.child_frame_id = child_frame_id
   t.transform.translation.x = matrix[0][3]
   t.transform.translation.y = matrix[1][3]
   t.transform.translation.z = matrix[2][3]
   t.transform.rotation.w = sqrt(
      max(0, 1 + matrix[0][0] + matrix[1][1] + matrix[2][2])) / 2.0
   t.transform.rotation.x = sqrt(
      max(0, 1 + matrix[0][0] - matrix[1][1] - matrix[2][2])) / 2.0
   t.transform.rotation.y = sqrt(
      max(0, 1 - matrix[0][0] + matrix[1][1] - matrix[2][2])) / 2.0
   t.transform.rotation.z = sqrt(
      max(0, 1 - matrix[0][0] - matrix[1][1] + matrix[2][2])) / 2.0
   t.transform.rotation.x = copysign(
      t.transform.rotation.x, matrix[2][1] - matrix[1][2])
   t.transform.rotation.y = copysign(
      t.transform.rotation.y, matrix[0][2] - matrix[2][0])
   t.transform.rotation.z = copysign(
      t.transform.rotation.z, matrix[1][0] - matrix[0][1])

   if to_ros_reference_frame:
      tr = t.transform.translation
      rot = t.transform.rotation
      tr.z, tr.y, tr.x = tr.y, -tr.x, -tr.z
      rot.z, rot.y, rot.x= rot.y, rot.x, rot.z

   return t
```

Listing 3.1: *The function for transforming the co-ordinate matrix retrieved from the HTC Publisher from the coordinate system used by the Vive and OpenVR to the co-ordinate system used by ROS so that the movement data can be accurately mapped. The function takes the HTC Vive data matrix and uses translation and rotation to generate the matrix corresponding to the ROS co-ordinate points.*

```
def listener():
   # Subscribe to topics to receive data from HTC publishers
   left_pose= rospy.Subscriber('/left_controller_as_posestamped', PoseStamped,
       left_pose_cb, queue_size=1)

   right_pose= rospy.Subscriber('/right_controller_as_posestamped', PoseStamped,
       right_pose_cb, queue_size=1)

   left_joy= rospy.Subscriber('/vive_left', Joy, left_joy_cb, queue_size=1)

   right_joy = rospy.Subscriber('/vive_right', Joy, right_joy_cb, queue_size=1)

   hmd_pose= rospy.Subscriber('/tf', TFMessage, pan_head, queue_size=1)
```

Listing 3.2: *The main function of the Baxter Subscriber application, the listener() function. When passed a specific message (the first parameter of the Subscriber() function) it will apply the desired function (the third parameter) to the message.*

The subscriber also makes use of several "callback" functions for the Joy messages, which are used to trigger the button commands talked about in Section 4.2.1. There was also an additional command implemented - a "pan mode" for better control of Baxter's head which is also what the callback for the TFMessage is related to, but that will be discussed in the Evaluation chapter.

The **constants.py** file is not a script and uses only a single function. It contains a number of predefined constants that are necessary for mapping the user to the robot. This file exists to hold these constants that may be shared across multiple other scripts as well as other values which are helpful to keep track of. Amongst these constants is the human-robot mapping ratio, which is used to accurately apply the data from the publisher to Baxter. This is used to fix the difference of origins in the system, as Baxter's origin is in his base frame and the Vive's HMD acts as the user's origin. The offset is found by finding the angle between Baxter's origin and his maximum movement range (the farthest that his arm can reach horizontally) and then using that angle to calculate the required offset to adjust the origin so that they match. A change to the user's arm measurement value will adjust the ratio accordingly, allowing quick changing of the ratio as needed.

$$\theta = \tan^{-1} \frac{ROBOT\_SHOULDER\_TO\_BASE}{ROBOT\_LIMBS\_OUTSTRETCHED} \tag{3.7}$$

$$user\_shoulder\_to\_base = tan(\theta) * HUMAN\_ARMS\_OUTSTRETCHED \tag{3.8}$$

$$offset = user\_shoulder\_to\_base + HUMAN\_SHOULDER\_TO\_HMD \tag{3.9}$$

Finally, the **main.py** script is the main loop of the application. It starts by initialising the Baxter Subscriber ROS node, then checks all the subscribers before enabling Baxter. This results in a lull period of no activity of roughly 10 seconds before it moves Baxter into a neutral position that is a suitable start point for mimicking human movement. It then runs the *move_to_goal()* function which takes whichever arm is currently being passed as a parameter and moves it to its end point using the Inverse Kinematics service (as long as the "roll mode" isn't currently active). This is the core script that has to be run to launch the application

3.4 Camera-HTC Application

The application that handles the video feed for the system is the main obstacle that the project has to overcome. ROS has built in camera control features that are designed to work with the cameras built into the robots. This lead to two major development hurdles:

- The simulator does not have any built in compatibility with the camera control functions and as such there is no way to test if the code will work without using the physical Baxter robot
- The Baxter robot only has a single eye camera, which means that in order to create a binocular vision system - which is necessary to fully achieve immersion - it is necessary to implement an external binocular camera into the system.

The second of these issues is relatively simple to solve and already had a potential workaround in the form of the ZED camera, a binocular camera used in previous attempts in this field. The first issue, however, would require coding in an entirely separate way of handling image and video feeds into the system at which point we would not be testing the original vision system.

While conducting the initial work on the system, it soon became apparant that incorporating the camera system as its own application would not be possible due to the limitations of the other components of the system. As mentioned previously SteamVR only allows interaction with a single game or application at a time and will close the initial application in favour of the newer application. In the previous study by Hew (2018), they proposed two potential solutions for this conundrum.

The first, and the less convenient of the two, was to rewrite the application using the Unity engine. Since many products developed for SteamVR use Unity or other similar engines, there is already plenty of documentation that could provide support on implementing the vision system. In addition, this would allow the whole system to be built as a single application rather than several and could have led to a simpler method of running the application i.e. running a single file in Unity compared to manually running the main scripts separately. However, this approach would mean that all development done to this point on the movement tracking and mapping applications would have to be scrapped and started from the ground up in Unity which would have resulted in a much more intense development cycle and less time to test and evaluate the system once it is completed, given the time constraints.

The other option put forward by Hew was to use the OpenGL library on Python to implement the ZED camera integration with the HTC Vive. This uses the official ROS-wrapper to individually access the topics for each of the camera's lenses and extract the image data. This solution was partially implemented by Hew but was abandoned due to unresolvable errors. It was decided that this partial implementation would provide a good starting point to work on and further develop as it did not require a complete restart of development.

However, an issue arose part way through the development cycle which forced a shift of scope and required a working solution for the camera in the simulator. The existing code that could be used as a base for these systems would only run on the proper hardware i.e. Baxter and the ZED camera. With both of those unavailable, a workaround had to be found for the workaround. Just as with the real world setup, the simulated Baxter had to have a binocular camera attached to his head in the form of a virtual Kinect. This wasn't an ideal fix, since the ROS camera controls for Baxter only work on the built-in cameras even in the simulator. This was solved by having the Baxter Subscriber node publish the video feeds as a ROS message and using the HTC Publisher node to subscribe to the feed which enabled a very basic feed. This had to be done by collecting a single frame at a time and adding it to a buffer before publishing it, which results in a laggy and unstable video feed that crashes most of the time.

3.5 Summary of Design and Implementation

The system architecture shown in this chapter was designed such that it would be possible to integrate all of the distinct technologies covered in Chapter 3 into a single Linux-based teleoperation system. This was initially built as three separate applications: a HTC Publisher application to collect and publish the input data by interfacing with SteamVR, a Baxter Subscriber application that takes the messages from the multiple topics and interfaces with Baxter using subscribers and a Camera-HTC Application to act as a pipeline that takes the captured video feed from the binocular camera and sends it to the Vive's HMD, sending the feed of each of the lenses to its respective eye.

However, this model required some reworking when it came to implementation as the bottleneck created by SteamVR's limitation of running a single application at a time forced the Camera-HTC Application to be reworked and divided across the other two applications (only one of which would directly interface with SteamVR). An attempt was made to act on a suggestion to rewrite the vision component in OpenGL but due to time constraints and the developer's lack of familiarity with OpenGL this idea was rejected in favour of a rather brute force workaround that only works some of the time. These issues highlight the difficulty in trying to create this kind of teleoperation system on a single Linux system due to the extreme lack of supported technologies and the poor support for those that are supported.

The next chapter will detail a number of tests that were done to evaluate the success of the system.

4 Evaluation

With the development of the teleoperation system covered in the previous chapter, this chapter will consist of the evaluation of its function and accuracy to determine its suitability and success of implementing a virtual reality interface for interfacing with Baxter.

4.1 Movement Tracking Accuracy

In order to determine how accurate the teleoperation system is and how appropriate it is for the task of interfacing with Baxter, it is useful to check both how quickly it can mimic the mapped movements and to what degree of accuracy it can do so. In addition, it is worth testing the reliability of the teleoperation system and in particular how consistent or inconsistent the problems with the vision system are.

In order to test the accuracy and speed of the mapping system, pose data was collected from the controllers while the user had their arms full outstretched in the X, Y and Z axes. This data was collected 10 times, with the time taken for the Baxter simulation to match the poses was also recorded. The mean and standard deviation of the three axes readings and the time was then calculated. Since the simulator doesn't run in real-time (the implications and effect of which is discussed later in Section 5.4), the time taken for each movement was recorded both as the simulated time and the real world timeframe and calculated separately using the average difference in timeflow putting the adjusted time at 1.67 times normal time.

Table 4.1: *A summary of the accuracy and response time results showing standard deviation*

	X co-ord value	Y co-ord value	Z co-ord value	Time (s)	Adjusted Time (s)
Mean	0.41653364	0.73992857	0.97065980	0.46	0.79
Standard Dev.	0.03205147	0.02896659	0.05242602	0.21	0.37

From the figures in the table we can see that the standard deviation for the coordinate values is close to zero, which tells us that the tracking system is consistently able to reproduce the desired mapping and poses with error small enough that it could possibly be disregarded. The standard deviation on the time isn't as small, but as these tests were run on the simulator instead of the Baxter robot it may just be that the latency within the simulator and teleoperation system is causing the variation in response time.

4.2 System Usability

During the development of the teleoperation system, a quirk within the system wad discovered when trying to map the movements of the user. The HTC Vive's tracking process maps the controller's co-ordinates using the HMD as a reference frame, which leads to an initially unexpected outcome of the HMD actively interfering with the tracking of the counter. Tilting and

turning the HMD shifts the position of the controllers and throws off all accuracy in the system, requiring a solution to be found.

A solution that was found by Hew (2018) in their project was the implementation of a "pan mode" for the robot similar to the "roll mode" for controlling the wrists. This prevents Baxter's head from rotating around the Z-axis until a button is pressed – assigned to the left Vive controller's menu button just as theirs was. This was implemented into the teleoperation system as a temporary fix until a better solution could be found, but due to time constraints it was included in the final build. While this does provide a partial fix, it is far from an effective one.

For one, the user is still forced to keep their head stationary while operating Baxter's limbs. This could provide discomfort to the user, who may already be struggling with operating the system for any length of time due to the weight of the HMD. It is also completely unreasonable to expect a human operator to remain completely still whilst using the system and as such some movement will be made which will throw the mapping of the arms off. In addition, the system will only update Baxter's head orientation when the "pan mode" button has been released, which is not only jarring for the user operating the system having to turn their head with their point of view not changing until they have stopped turning and released the button at which point it snaps to the new angle. This workaround also had the unforeseen effect of crashing the visual feed whenever it was activated.

There do exist other methods that could potentially resolve this issue, such as decoupling the HMD and controllers, but due to time constraints it wasn't possible to implement a better fix than this.

4.3 Unresolved Issues

While most of the problems encountered during the course of the project were able to either be solved or worked around, there were several difficulties that proved either too difficult to solve or too important to ignore. This section will detail the most prominent of these, as well as any success that was made in solving them.

Our initial plan was to develop the system on a laptop running Linux for convenience, which would allow work to be done outside of the Baxter lab and increasing the ease of production. However, Steam proved far more difficult to install than we expected due to various inconsistencies between packages that were updated between releases. In addition to this, the ROS package and Baxter simulator also encountered similar issues when installing which we believe was due to the open-source nature of ROS and Gazebo leading to software packages that could easily "tangle themselves up" for lack of a better term. By the time we came to the conclusion that developing on the laptop was too much work, two months had passed without any code having been properly run and tested. There is a good chance that had we thrown in the towel with regards to development on this laptop earlier, more work could have been done on the project.

The layout of the Baxter lab proved an issue when it came to setting up the HTC Vive. As detailed in Figure 2.5, the intended setup is to have both sensors in opposite corners of the room in order to provide as close to complete coverage with the sensors as possible. This wasn't possible in the Baxter lab, as two of the four walls of the room are made of glass and therefore are unsuitable for attaching the wall mounts to. The mounts can be used to attach the base stations to C-stands or similar pole-based stands, which was not possible since the mounts were missing from the HTC Vive's box. As a result of this, an alternative setup had to be found. It was discovered that the base stations would work as long as they had line of sight with one another, and so the setup detailed in Figure 4.1 was settled on. It is unknown if this setup is the cause of any of the technical issues experienced during the course of the project, but it can be assumed that some of the loss

of tracking (especially when it occurred raising the controllers above the user's head or above 177cm) was due to this unconventional setup.

Figure 4.1: *The figure above shows the finalized room layout that was settled upon during the project. With the base stations placed at an angle to the edge of the desk so that they have line of sight with each other, the HTC Vive appears to work as normal. Of important note is that the base stations sit at the edge of the desk, as otherwise the desk blocks any signals that would occur past it.*

Due to the change of scope mentioned in previous sections, other issues were discovered relating to running the teleoperation system on the simulation which made the later stages of development difficult. The most notable of these is that the simulator doesn't run in real-time, instead varying between 20% and 60% slower than real-time. This initially didn't appear to be of any concern, instead being a simple quirk that came with working on the simulator. This changed when we discovered that the difference in time flow was at least partially responsible for many of the inaccuracies with the system. The seemingly random shaking, which we believed was a result of an overly high sample rate until we couldn't do more than reduce it to a somewhat acceptable level, was discovered to be due to the system trying to simulate Baxter moving to the mapped pose as intended but amplifying any small movements made by the system and displaying them as violent shaking.

4.4 Summary of Evaluation

By testing the accuracy of the teleoperation system in terms of its accuracy across repeated tests, it was determined that the system architecture used to process the input from the Vive was implemented appropriately as it produced results that aligned with the expected outcomes and responded quickly enough to satisfy our needs. However, the system is not entirely reliable in replicating poses due to the limitations of Baxter's joints.

A "pan mode" was built into the teleoperation system as an attempt to rectify the skewing of the reference frames caused by shifting the Vive's HMD. It does provide a partial fix in preventing both the head and controllers moving simultaneously, but it was a far from ideal solution as it

locked Baxter's head and point of view while the pan mode is not enabled. It also forced the user to keep their head still and had the consistent side effect of crashing the visual feed whenever it was activated.

There were several additional problems during the course of the project that were unable to be rectified due to either time restraints or changing of priorities. These include being unable to build the teleoperation system on a Linux laptop for ease of development and having to find an alternate setup for the Baxter lab due to an inability to use the intended setup.

5 Conclusion

5.1 Achievements

The goal of this project was to develop a teleoperation system to drive the Baxter research robot using consumer grade hardware and low cost graphics software that would run on a single Linux machine. The final build of the system utilises the HTC Vive to interface with Baxter through two applications that utilise the OpenVR API to communicate with SteamVR at runtime to allow both input and output from the HTC Vive. This section will detail the achievements of this project.

The system works through the use of the ROS publisher and subscriber system which allowed the inputs of the HTC Vive controllers to be retrieved and processed so that the instructions that could be mapped to Baxter. This led to successful accurate mapping of the human user's movements to the robot allowing the user to controll Baxter's arms, grippers and - to a limited extent - its head. With the confirmation provided by testing the accuracy of the system.

The system was also altered in order to operate on the Baxter Gazebo situation after unforeseen circumstances took the physical Baxter robot offline for the latter stages of the development period. Although this was a major hurdle in the project and forced a shift in scope with fairly little time to work around it, the final build of the system does have a working mapping system that should translate to the Baxter robot with few difficulties.

5.2 Further Work

While the current teleoperation system is improved over previous attempts, there is still work that can be done to improve even further.

Firstly, the vision system only functions in a minimal sense - it suffers from up to 2 seconds of lag on average when it is running, which can be difficult to work with and can induce cybersickness in the user if they are susceptible to it. In addition, getting the system to work can also be a challenge due to near constant crashing for reasons that couldn't be determined due to the time scale. The only guaranteed crashes that could be identified was when an attempt was made to capture the video feed using outside software and when the "pan mode" mentioned in Section 5.2 was activated. This would appear to be due to the method through which the images are transferred - they are essentially shuffled frame by frame through the handler file, causing the extreme lag due to having to "prep" the footage before transmitting it (or else suffer single digit framerates while operating and raising the chance of cybersickness) which could lag to the point of stalling when the outside program is trying to observe the same frames. Ultimately, more research needs to be done into handling the footage before a fix can be settled on.

Furthermore, work should also be done into looking for ways around the restrictions placed on the system due to its reliance on Linux. Most other virtual reality systems have more robust documentation and support on Windows and MacOS compared to on Linux, so it may be possible to discover a workaround for some of the various other issues and bugs that arose during the

course of the project. This task of looking into various restrictions placed on the implementation by reliance on Linux could also be further extended to looking for alternative hardware and software to trial the system with. For example, a more humanoid robot such as NAO could provided more accurate results as to the quality of the system or expand the movement tracking of the user to use full-body tracking as opposed to room-scale tracking.

It may also be wise to take another look at this particular type of project and whether or not such a task will pay off. The primary goal behind creating a working teleoperation interface on a single Linux machine is to cut back on costs - using one computer instead of two means only half the expenses for the powerful graphics cards and other (less relevant) components for the machines. The trade-off, however, appears to be both in the quality of the system and the ease of development. This project and both of its predecessors by Harper and Hew have slowly made progress towards a working solution to this problem over three years of iterative work, building off of what worked and what didn't. Even with this project being the most complete version thus far, it is still far from being in a presentable state that could be used for Open Day Demonstrations or training by example due to the barely functioning visual system mentioned above. On the other hand, the system produced by Vanik (2017) made use of a two machine setup that achieved success far beyond what the one-machine builds did. It may be worth reevaluating how much would be gotten out of this type of setup and perhaps shelving the idea for a few years in order for the Linux support to become up to scratch. A point could be made, however, that it may be worth plowing onwards with this idea in case no such advances in Linux support occur.

Another possible future step would be to approach the teleoperation interface from the context of human-computer interation and the psychology of the field. This interface is far from traditional, which provides a ripe oppurtunity to explore how different users experience using the teleoperation interface. There is already some research into cybersickness which could be used as a springing off point, looking into trying to tweak the system to limit such symptoms where possible such as stabilising the frame rate. There is also the reports of existing users to take into account, who all report a strange sense of wonder when they used the interface for the first time. They all mention the disconnect of putting on the headset and then seeing Baxter's grippers where their hand's aught to be. This could be ripe for exploration, even as a possible joint Computer Science and Psychology endeavour.

A Appendices

A.1 Co-ordinate Mapping Equation and Functions

These equations are the basis of the function A.1

$$X_{ROS} = -Z_{VIVE} \tag{A.1}$$

$$Y_{ROS} = -X_{VIVE} \tag{A.2}$$

$$Z_{ROS} = +Y_{VIVE} \tag{A.3}$$

A.2 Rotation Mapping Equations

$$Roll_{BAXTER} = -Yaw_{VIVE} \tag{A.4}$$

$$Pitch_{BAXTER} = -Pitch_{VIVE} + 90° \tag{A.5}$$

$$Yaw_{BAXTER} = -Roll_{VIVE} \tag{A.6}$$

A.3 Accuracy Tracking Results

	X co-ord value	Y co-ord value	Z co-ord value	Time (s)	Adjusted Time (s)
1	0.42338479	0.76459323	1.03123675	0.5	0.9
2	0.47024324	0.72104926	1.05635596	0.7	1.2
3	0.41204488	0.72378963	1.03091739	0.3	0.5
4	0.39948272	0.76222679	0.97632635	0.4	0.7
5	0.37862113	0.71953548	0.93192274	0.2	0.3
6	0.40923321	0.73114735	0.91554509	0.2	0.3
7	0.40157133	0.70398249	0.91471325	0.9	1.5
8	0.47246236	0.76348503	0.92367012	0.5	0.9
9	0.38247378	0.71568406	0.95725683	0.4	0.7
10	0.41581900	0.79379235	0.96865354	0.5	0.9
Mean	0.41653364	0.73992857	0.97065980	0.46	0.79
Standard Dev.	0.03205147	0.02896659	0.05242602	0.21	0.37
Co. of Variance	0.07694811	0.03914781	0.05401070	0.47	0.48

```
def from_matrix_to_transform(matrix, stamp, frame_id, child_frame_id,
                     to_ros_reference_frame=True):
   t = TransformStamped()
   t.header.stamp = stamp
   t.header.frame_id = frame_id
   t.child_frame_id = child_frame_id
   t.transform.translation.x = matrix[0][3]
   t.transform.translation.y = matrix[1][3]
   t.transform.translation.z = matrix[2][3]
   t.transform.rotation.w = sqrt(
      max(0, 1 + matrix[0][0] + matrix[1][1] + matrix[2][2])) / 2.0
   t.transform.rotation.x = sqrt(
      max(0, 1 + matrix[0][0] - matrix[1][1] - matrix[2][2])) / 2.0
   t.transform.rotation.y = sqrt(
      max(0, 1 - matrix[0][0] + matrix[1][1] - matrix[2][2])) / 2.0
   t.transform.rotation.z = sqrt(
      max(0, 1 - matrix[0][0] - matrix[1][1] + matrix[2][2])) / 2.0
   t.transform.rotation.x = copysign(
      t.transform.rotation.x, matrix[2][1] - matrix[1][2])
   t.transform.rotation.y = copysign(
      t.transform.rotation.y, matrix[0][2] - matrix[2][0])
   t.transform.rotation.z = copysign(
      t.transform.rotation.z, matrix[1][0] - matrix[0][1])

   if to_ros_reference_frame:
      tr = t.transform.translation
      rot = t.transform.rotation
      tr.z, tr.y, tr.x = tr.y, -tr.x, -tr.z
      rot.z, rot.y, rot.x= rot.y, rot.x, rot.z

   return t
```

Listing A.1: *The function for transforming the co-ordinate matrix retrieved from the HTC Publisher from the coordinate system used by the Vive and OpenVR to the co-ordinate system used by ROS so that the movement data can be accurately mapped.*

Bibliography

C. C. Bracken and P. Skalski. *Immersed in media: Telepresence in everyday life*. Routeledge, 2010.

C. Hew. Teleoperation of baxter using the htc vive on a single linux system. 2018. MSci Dissertation, University of Glasgow.

H.-H. Hsu, Y. Chiou, Y.-R. Chen, and T. Shih. Using kinect to develop a smart meeting room. pages 410–415, 09 2013. ISBN 978-1-4799-2509-4. doi: 10.1109/NBiS.2013.66.

J. I. Lipton, A. J. Fay, and D. Rus. Baxter's homunculus: Virtual reality spaces for teleoperation in manufacturing. *IEEE Robotics and Automation Letters*, 3(1):179–186, Jan 2018. ISSN 2377-3766. doi: 10.1109/LRA.2017.2737046.

D. C. Niehorster, L. Li, and M. Lappe. The accuracy and precision of position and orientation tracking in the htc vive virtual reality system for scientific research. *i-Perception*, 8(3): 2041669517708205, 2017. doi: 10.1177/2041669517708205. URL `https://doi.org/10.1177/2041669517708205`.

J. Siebert and D. M. Shafer. Control mapping in virtual reality: effects on spatial presence and controller naturalness. 2017. URL `https://link.springer.com/article/10.1007/s10055-017-0316-1`.

P. Vanik. Teleoperation of the baxter robot using oculus touch and zed camera. 2017. Ph.D Dissertation, University of Glasgow.

N. G. Vinson, J.-F. Lapointe, A. Parush, and S. Roberts. Cybersickness induced by desktop virtual reality. In *Proceedings of Graphics Interface 2012*, GI '12, pages 69–75, Toronto, Ont., Canada, Canada, 2012. Canadian Information Processing Society. ISBN 978-1-4503-1420-6. URL `http://dl.acm.org/citation.cfm?id=2305276.2305288`.

D. Weibel and B. Wissmath. Immersion in computer games: The role of spatial presence and flow. *Int. J. Comput. Games Technol.*, 2011:6:6–6:6, Jan. 2011. ISSN 1687-7047. doi: 10.1155/2011/282345. URL `http://dx.doi.org/10.1155/2011/282345`.

Y. Xu, C. Yang, P. Liang, L. Zhao, and Z. Li. Development of a hybrid motion capture method using myo armband with application to teleoperation. pages 1179–1184, 08 2016. doi: 10.1109/ICMA.2016.7558729.