Electronics and Computer Science
Faculty of Physical Sciences and Engineering
University of Southampton

Adam Melvin

April 2018

Using Data Abstraction and Inter-Frame Interpolation for Low Data Rate Communication Between a 3D Camera and VR Headset

Project supervisor: Klaus-Peter Zauner
Second examiner: Sheng Chen

A project report submitted for the award of
MEng Electronic Engineering

UNIVERSITY OF SOUTHAMPTON

Abstract

Faculty of Physical Sciences and Engineering
Electronics and Computer Science

MEng Electronic Engineering

by Adam Melvin

The application of Virtual Reality (VR) to telerobotics is a current area of study in industry due to a desire for the increased spacial awareness VR provides. However, attempts to implement such a system using standard teleoperation techniques result in sub-par performance and an uncomfortable experience for the user; the benefits of a VR based system are entirely eliminated. The system proposed by this project incorporates elements of data abstraction and inter-frame interpolation to produce abstractions of the environment with lower performance requirements, as opposed to the standard approach of aiming for the presentation of an exact replica.

Contents

List of Figures

Acknowledgements

I would like to thank Klaus-Peter Zauner and all the people who have had to put up with being constantly on camera for their patience. I would also like to thank Tom Darlison for allowing me access to his photographs.

Chapter 1

Introduction

There are many areas of science and engineering that require the observation of, and interaction with, environments not suitable for human beings. These environments are instead observed using teleoperated robots, potentially removing the need to put people in any danger. However, a challenge presented by telerobotics is providing the operator sufficient information about the robot's surroundings to give them a feeling of presence [1] within the space; this is essential for effective manoeuvring and interaction.

Virtual Reality (VR) is a technology that has proven itself to be able to provide the user with unparalleled presence within a virtual space- comparable to presence within a real, physical space [2, 3]. To be able to incorporate VR into teleoperation is therefore desirable. While this has been successfully attempted for the purpose of controlling robots within an already mapped space [4], there has been much less success in exploring an unknown space through a VR interface. This is due to the high frame rate and low latency required to prevent motion sickness while in a VR environment.

It's widely accepted that for a VR application to not cause motion sickness and headaches due to frame rate, it must maintain at least 90 frames per second (fps) [5]; a minimum of 60 fps can also be acceptable [6], but generally only for applications with little motion or when used by people with lower susceptibility to motion sickness. Unfortunately, to transmit 90 fps from a stereo camera rig (two images are required to perceive 3D) to the computer running the VR application has very high bandwidth

requirements. Also, a major factor in providing presence to the user in VR is their ability to look around the space independently. This can be achieved by mounting the stereo camera rig on a gimble, however to build a gimble that is able to track the angle of the user's head accurately and with low latency is both expensive and challenging [7]; if not implemented perfectly the user would be more likely to suffer sickness and dissociation from the space than if the gimble was not used at all.

The aim of this project is to design and implement a VR based teleoperation system that utilises data abstraction and inter-frame interpolation to minimise the outlined technical issues, providing increased comfort and therefore presence to the user than otherwise possible. Data abstraction would be used in the robot to reduce each image down to its most essential features, reducing its size and therefore the required data rate significantly. Each image pair would then transmitted to a server and combined into a single 3D map of the space that could be looked around freely through the VR headset. As the camera feed would be viewed as a 3D environment rather than directly as images, the headset could run at the full 90fps even if the environment is updating at a much slower rate. As previously suggested, a camera gimble would track the movement of the headset, but it would not have to be very accurate as it would only updating the 3D map and not affect the headset directly.

The system would be realised using off-the-shelf VR equipment, a camera gimble adapted from one produced by previous students [8], and a simple rover that I would also build.

Chapter 2

Background

2.1 Virtual Reality

The term Virtual Reality (VR) refers to the generation of a 3D environment that can be interacted with by a user in a realistic fashion, with the aim of immersing the user in the environment as if it were the real world [9]. While there are a large array of systems that can be considered VR, whenever the term is used in this report it is only referring to the head-mounted display (HMD) systems that have become popular in recent years with the release of the Oculus Rift [10] and the HTC Vive [11] (Figure 2.1); these are both consumer grade systems that are aimed at the immersive gaming market.

FIGURE 2.1: HTC Vive. Pictures of the Vive headset, reproduced from [11].

HMD based systems display different images for each eye to provide the user with a sense of depth within the 3D environment, making the headset effectively operate like a pair of binoculars into the virtual world. The headset is also tracked in 3D space, and this movement translated into the 3D environment with very low latency. These features, among others, are all implemented with the aim of providing the user with presence within the virtual space that is comparable to observing the real world.

Due to its availability at the University of Southampton and in my own home, the HTC Vive was used as the VR device in the implementation of the teleoperations system discussed in this report.

2.2 Telerobotics

As discussed in the introduction, a telerobot is a robot controlled from a distance by a human operator [12]. Telerobots are typically developed to undertake activities within environments that are too dangerous or costly for humans to work in, an example being the extensive telerobotics research at NASA for tasks such as deep-space exploration [13].

2.2.1 VR in Telerobotics

The use of HMDs in teleoperations is not a new concept; NASA's Robonaut 2 was sent to the International Space Station in 2011 and can be controlled through a headset that displays the output of the robot's head cameras [14], and flying drones by First Person View (FPV), an analogue video feed transmitted over radio to a HMD, is very popular [15]. However, these systems are either incredibly expensive (Robonaut 2 is worth millions of dollars) or very limited (FPV systems send one, low quality, video stream over a short distance), and all have the motion sickness issues discussed in the introduction. While stereo camera FPV systems have been developed, so the user has depth perception and better presence in the drone's view, the motion sickness problem remains the major drawback of HMD based teleoperations systems [16].

As previously established, motion sickness in VR is mitigated through high frame rates and low latency. However, most VR based teleoperations systems currently available are direct VR systems, so they display the video feeds produced from the device's cameras directly in the headset. This entirely ties the frame rate and latency of the headset to the capabilities of the video transmission system, and only the most expensive and complicated systems will meet the strict requirements for comfortable VR.

An alternate option to a direct system is an indirect system. This is one in which the video feed is abstracted from the headset in some way in the hope of providing improved comfort and awareness. Indirect systems can come in a variety of forms, such as a virtual control room with the video feed on a virtual screen [17], or a 3D map generated from a multi-line LiDAR and IMU [18] (Figure 2.2). These examples show the potential of indirect systems as a solution to VR based teleoperations, though the research into this field is currently minimal; this project aims to expand on this research with the development of its novel system.

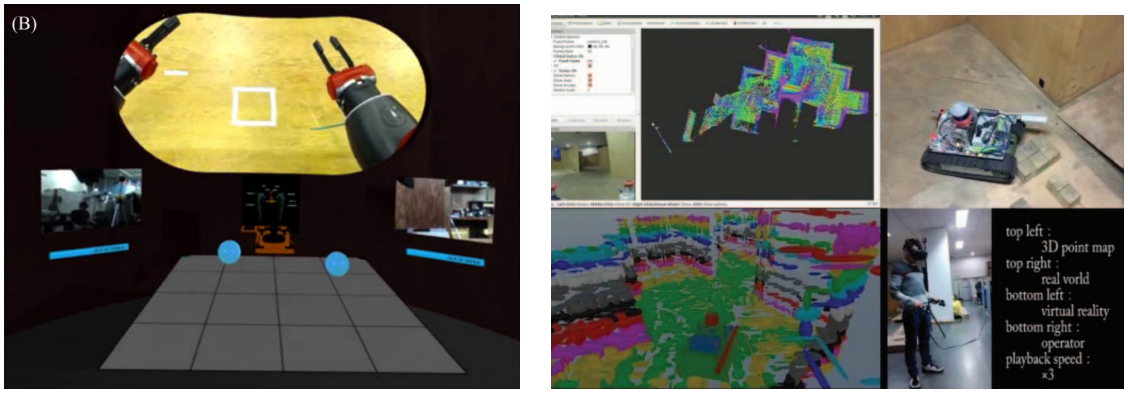


FIGURE 2.2: Indirect VR teleoperations examples. A virtual control room based system (left) and a LiDAR 3D map based system (right), reproduced from [17] and [18].

2.3 Data Abstraction

"Data abstraction" is the phrase that will be used in this report to describe the act of reducing an image down to only its most essential elements. It is similar in concept to an artist sketching a scene instead of attempting a full drawing, and is

comparable to data compression as the aim is also to reduce the file size of the image, however data abstraction takes a very different approach to solving the problem than standard compression algorithms.

Data compression is the storing of information using a more space efficient encoding [19]. While some information is lost during lossy compression, the aim regardless of the algorithm used is to retain as much of the original information as possible. In contrast, the aim when utilising data abstraction is to discard all the information that is unnecessary to fulfilling the image's purpose. For example, if all that is required of an image is that basic shapes can be identified, then only the information on the boundaries of the shapes is necessary; the rest of the image can be discarded. An implementation of data abstraction can be seen in Figure 2.3.

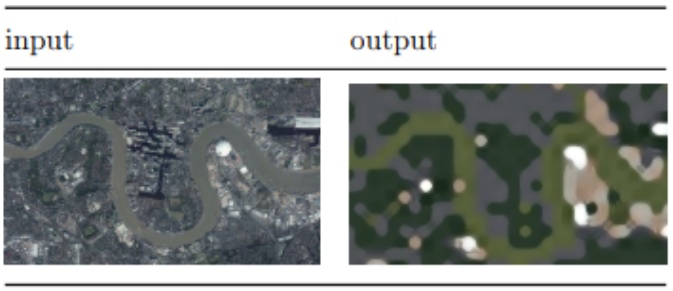


FIGURE 2.3: Data Abstraction Example. This is the abstraction of an aerial photograph of London, reproduced from [20]

2.4 Inter-Frame Interpolation

Inter-frame interpolation is the generation of intermediate frames in a video feed to increase its frame rate. This is typically done using estimations of optical flow to interpolate frames within a pre-recorded sequence [21], though there are methods that can achieve real-time video processing when provided with powerful enough hardware [22]. In this report, the term inter-frame interpolation refers to the increase in frame rate provided by the conversion of the video feed into a 3D map, as in effect we are interpolating the <30fps video feed up to 90fps in the VR headset.

2.5 Sobol Sequences

Sobol sequences are quasi-random sequences that were introduced to aid in approximating integrals. The aim is to form a sequence of points that are evenly spread across an S-dimensional unit cube [23]. This provides a much more even spread of points across the chosen space than can be produced from a pseudo-random number source (Figure 2.4). The code used in this project to produce these sequences was created by Leonhard Grünschloß [24].

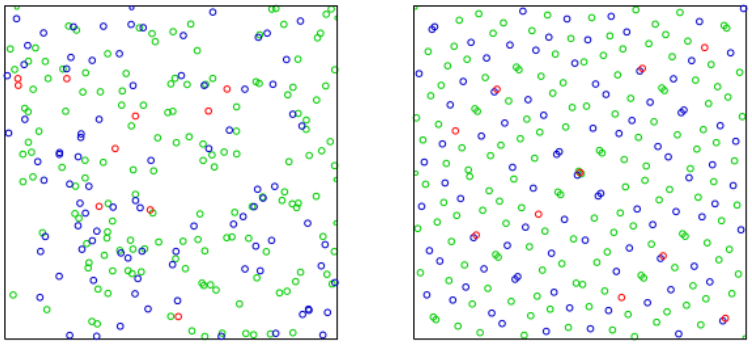


FIGURE 2.4: Comparison of pseudo-random and quasi-random sequences. 256 points from a pseudo-random generator (left) and 256 points from a Sobol sequence (right), reproduced from [25].

2.6 Computer Vision

Computer vision is the automatic analysis of images and extraction of the useful information they contain [26]. A raw image is simply a large matrix of colour values, so for a computer to take action based on the contents of an image it must be able to recognise features using analysis of this data. Doing so involves many different techniques such as statistical pattern classification and geometric modelling [27]. All computer vision methods in this project are implemented using the OpenCV libraries, and the example programs provided with them used as starting points for development [28].

2.6.1 Edge Detection

When attempting to recognise the features of an image, knowing the locations of the edges of objects within the scene is often very useful. An edge is defined as a significant local change in intensity, usually due to a discontinuity in either the intensity or its first derivative [29]. There are many algorithms available that will detect the edges of an image from the locations of these discontinuities. When the most popular algorithms (Laplacian of Gaussian, Robert, Prewitt, Sobel, and Canny) are compared [30], the most effective in almost all scenarios is Canny edge detection [31], therefore this is the algorithm utilised in this project. Canny edge detection is demonstrated in Figure 2.5.

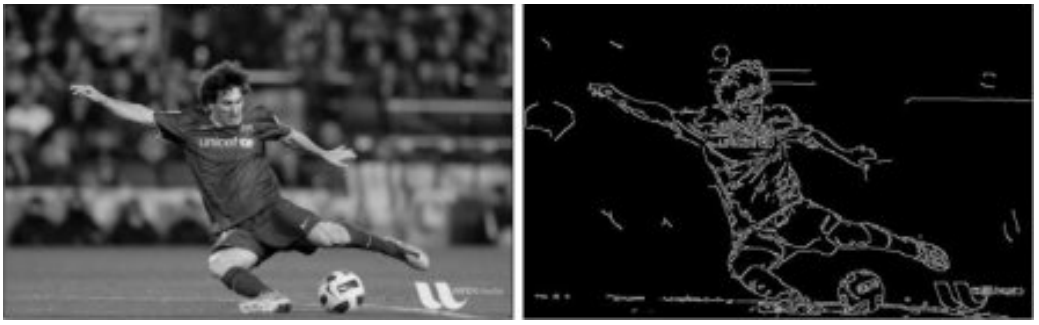


FIGURE 2.5: Canny Edge Detection Example. Simple edge detection program applied to a fairly detailed photo of Messi, to demonstrate its effectiveness even with more complex images. Figure taken from an OpenCV Canny tutorial [32].

2.6.2 Flood Fill

Flood fill algorithms determine the area connected to a given cell (the seed point) in a multi-dimensional array that have similar intensity values for the purpose of filling them with a chosen colour [33]. This is a technique that is not only useful in image processing, but also for many other fields such as in passive acoustic monitoring where finding the area connected to a given node can be useful as part of tracking in 4D space (x,y,z,time) [34]. A demonstration of flood fill has been presented in Figure 2.6.

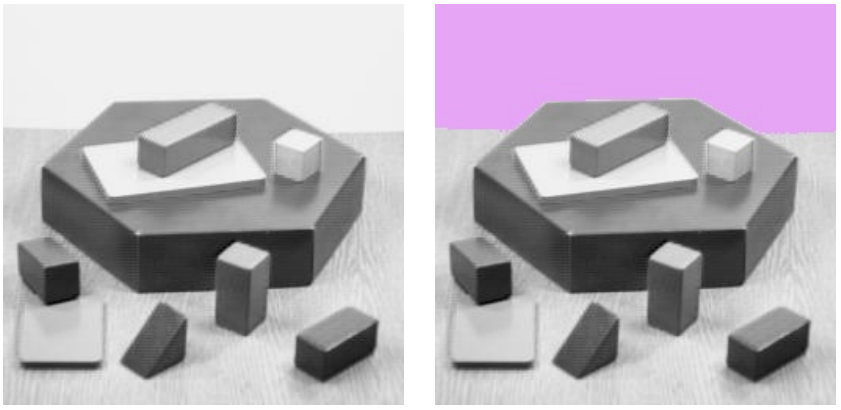


FIGURE 2.6: Example of Flood Fill. The original image (left) was provided by OpenCV. The right image is the result of flood filling from the top left corner.

2.6.3 Depth Mapping

The main component in the human brain's perception of 3D is the identification of disparity between the locations of objects in the 2D images being produced by our eyes [35]. The greater the difference in the horizontal placement of an object between the images, the closer the object to the observer. This technique can be used in computer vision to produce depth/disparity maps. Depth maps display differences in depth as a gradient from white to black (Figure 2.7), and can be produced using a variety of difference algorithms. The most common are block matching algorithms, which use simple geometry and the matching of blocks of pixels horizontally in the 2 images to calculate depth [36]. For these algorithms to locate the same object in different places in the 2 images, the cameras taking them must be calibrated to rectify any distortion due to the lenses [37] or discrepancies in the mounting that would cause them to be out of line [38].

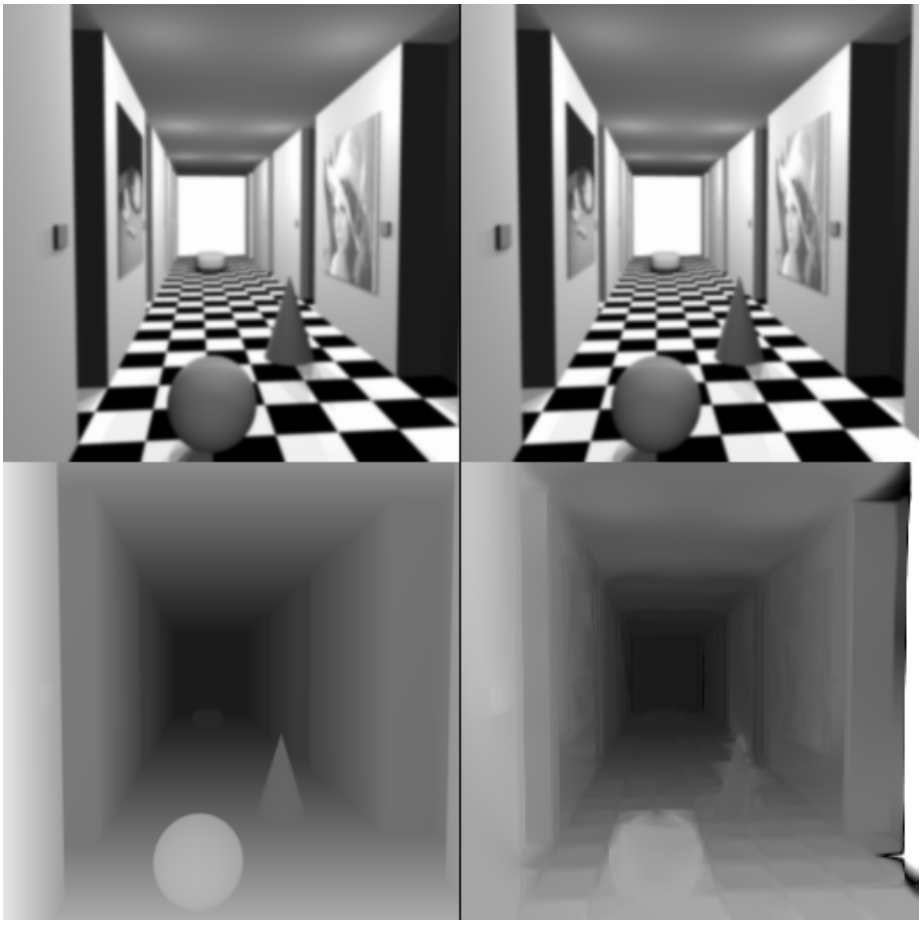


FIGURE 2.7: Depth Mapping Example. A stereo pair of images, with their exact disparity map bottom left and a disparity map produced by a dense disparity map estimation algorithm bottom right- reproduced from [39].

2.7 Wireless Communication

The different methods of wireless communication available for a telerobotics system have varying requirements and benefits. For instance, the analogue video transmission found in FPV drone flying (as mentioned in Section 2.2.1) is low latency at the expensive of video quality, range, and communication outside of line of sight. This is fine for flying a drone from the ground just below it, but would not be adequate for controlling a telerobot from a VR headset in a completely different room as is the aim of this project. A better choice of communication method would be over Wi-Fi, as a Wi-Fi router can easily allow communication across a whole building (or in the

case of university Wi-Fi - a whole campus). The trade off is an acceptable increase in latency.

The two main protocols available for Wi-Fi communication are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). TCP is a connection-oriented protocol, so uses handshaking to guarantee reliable and ordered transfer of data [40]. The trade off to this is that the handshaking slows down the process of communication. If the priority of a system is speed over packet reliability and ordering, UDP may be a better choice. There is no checking in UDP, so there is no guarantee that the packets will arrive but they can be sent with minimal overheads [41]. The chosen protocol for this system is UDP, as images not arriving in the correct order and the ocassional loss of packets would not have a significant effect on the functionality of the system in comparison to the usability improvement a reduction in latency provides.

Chapter 3

System Overview

As can be seen in Figure 3.2, the system consists of 2 platforms- A server that runs a VR environment and reads user input, and a rover platform that is controlled from said environment and supplies the abstracted images the environment is built from. The rover is a simple drivable platform with a stereo camera gimble mounted on it (Figure 3.1), and is the subject of Chapter 5. The server is a powerful PC running Windows 10 and a HTC Vive. The design of the program the server runs is the subject of Chapter 6. The data abstraction algorithm the system uses (in the "Data Abstraction" and "Coloured Abstraction Construction" blocks of Figure 3.2) is novel, so its design and development is initially discussed in isolation in Chapter 4 and then its application within the system addressed in Chapter 5. Finally, the full system will be evaluated in Chapter 7.

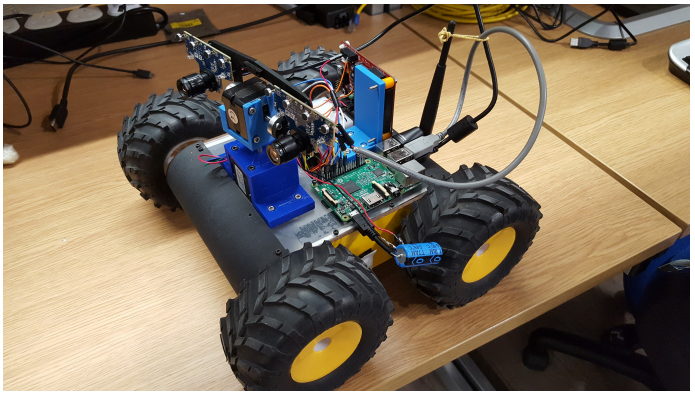


FIGURE 3.1: Rover Picture.

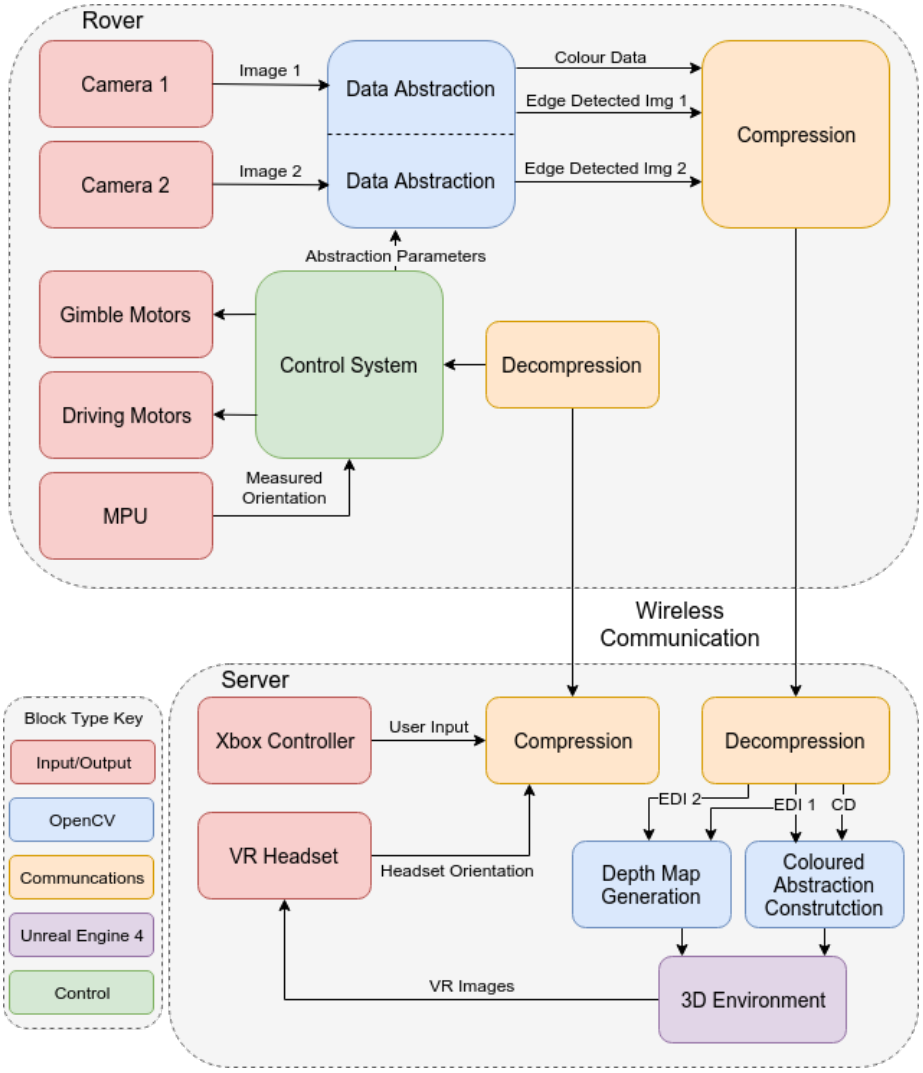


FIGURE 3.2: System Overview Block Diagram.

Chapter 4

Abstraction Algorithm Development

In this implementation of data abstraction, the aim is to reduce images down to only the boundaries of the objects in the scene and then fill the spaces between theses boundaries with block colours based on the original image. Therefore the general process of the design is:

1. Use edge detection on the image, presenting the boundaries of the scene as white lines and the rest as black.
2. Divide up the original image into sections that can reasonably be averaged into a single colour, defined by the boundaries produced by the edge detection or otherwise.
3. Find the average colours (or reasonable alternatives) for these sections.
4. Flood fill the spaces in the edge detection output with the average colours of the original image.

While this section will be presented in the context of the entire process occurring on a single computer, when incorporated into the final system points 1-3 are implemented on the rover and point 4 is implemented on the server ("Coloured Abstraction Constrution" on Figure 3.2). It is also worth noting that this chapter is concerned

with the algorithm's ability to produce recognisable abstractions under reasonable resource constraints- the file sizes it is capable of producing will be covered as part of the discussion of the full system's communications protocol later on in Section 5.2.

4.1 Edge Detection

Canny edge detection was implemented using the Canny function provided by OpenCV. The sequence of processes implemented to support the algorithm in producing high quality edges are shown in Figure 4.1.

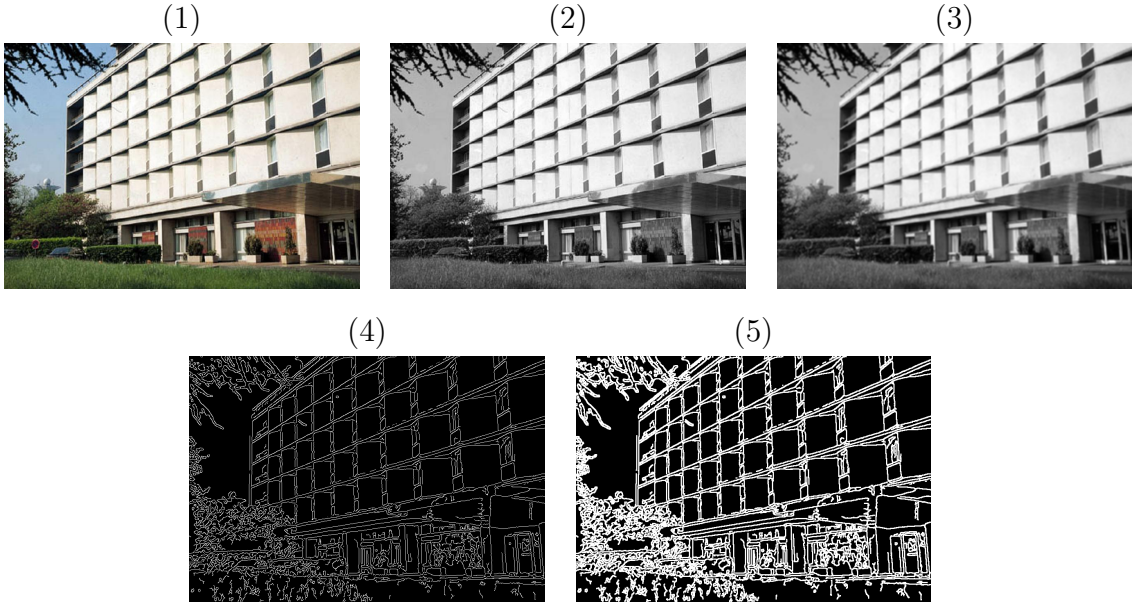


FIGURE 4.1: (1) the original image, provided by OpenCV, (2) post grayscaling, (3) post blurring, (4) post edge detection, and (5) post dilation.

The image is first made grayscale, as Canny detects large changes in light intensity and not colour. The image is then blurred to remove any unnecessary edges and noise that Canny may pick up. The edge detection is applied, producing white lines, representing the edges, on a black background. The output of the edge detection is finally dilated to make the lines thicker and bridge the gaps between the lines that are very close together. This is done to make the image more cartoon-like and more generally aesthetically pleasing, reduce the number of lines produced by areas that

are dense with detail such as hair and foliage, and to bridge the gaps between lines that are close together, increasing the likelihood of defined shapes being created that can be easily flood filled later.

4.2 Flood Filling

Flood filling was chosen as the method for applying colour to the Canny output image, as it is an effective method for fill spaces of unknown size and shape that are defined by high contrast boundaries; using it makes dividing up the image into sections unnecessary. To use the OpenCV flood fill function you must provide a seed point to start flooding from, a colour to fill with, and parameters for the filling itself (unchanged from the defaults provided by the OpenCV documentation [42]).

4.2.1 Seed Point

Finding the points to flood fill from is a challenge, as each image will have a different number of spaces to be filled and the spaces can be anywhere. Three different methods were attempted to solve the problem. The first was an attempt to use OpenCV's contour functionality to turn the lines into a set of contours and use the centre of mass of each contour as the seed point. However, this was unusable due to high resource requirements. The method is detailed in Appendix C.

Although it would by ideal to aim to flood fill from the centre point of each space, it is only necessary if you intend to be selective about which pixels are being used as seed points.. It is possible to instead iterate through the whole image and flood fill from every pixel found that is not part of a line or an already filled space. This method is more effective at filling every space than the previous, and is also less resource intensive. However, if presented with a complicated environment with many spaces to flood fill it must fill every single one, leading to unacceptable drops in frame rate.

A simple solution to the performance issues caused by complex images would be to set a maximum number of times flood fill can be used per image. However, if this is done the seed points can no longer be selected by iterating through the whole image, as the presence of many small spaces at the top of an image would lead to larger,

more important spaces not being filled at the bottom. The solution is to select a set number of points quasi-randomly across the image using Sobol sequencing (explained in Section 2.5). Although a certain number of points will land on lines and therefore not be used, if there are enough points then all the important spaces are filled without serious impact on performance. Also, with this method performance is not affected by the complexity of the image, however more complicated images will be processed with many of the more dense areas unfilled (Figure 4.2). For these reasons, the seed points are selected using this method in the current build.

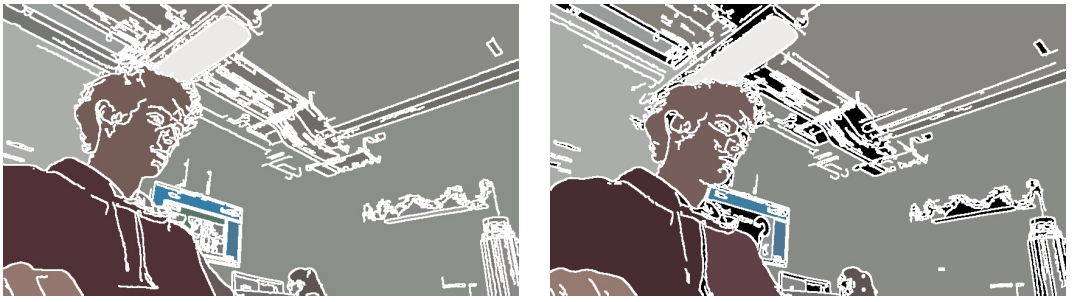


FIGURE 4.2: Comparison of brute force and Sobol seed point generation. It can be clearly seen that the brute force method (left) accurately fills every space in the image, whereas using Sobol results in many unfilled spaces. However, Sobol is higher performance with a frame rate of 17.14fps, compared to 8.57fps using brute force.[REDO FPS TESTS]

4.2.2 Fill Colour

Three different methods were considered for finding the colours that the Canny output should be flood filled with. All three methods are valid solutions, but present different ratios between accuracy and resource requirements.

While filling the abstracted image with the average colours present within the input is preferable, it is not essential to the project; provided that the objects in the scene are still recognisable, they don't have to be exactly the right colour. For this reason it would be acceptable to not find the average colour of the area being flood filled at all, and instead simply use the colour of the seed point. This is a very fast method, however produces incredibly inconsistent colours between images. This is because there can be a wide spectrum of colour across a single surface even within the threshold of Canny edge detection, and the Sobol sequence will sample from

a different point each time producing spaces that flicker between a wide range of colours.

The consistency of the previous method can be improved substantially with minimal impact on performance by taking an average of the colour within the area of a small circle around the seed point, rather than just the colour of that one point. This significantly improves the consistency between images, however introduces the problem of incorporating pixels from outside the space to be filled. This is due to the quasi-random points often being so close to the edge of the space that the averaging circle crosses the edge slighty and averages using part of a neighboring space. Therefore, the size of the circle must be carefully selected to balance the benefits of increasing size (more consistency when the seed point is further from the edge) and the benefits of decreasing size (more consistency when the seed point is closer to the edge).

The OpenCV flood fill function provides the ability to fill a blank mask using the boundaries defined by a different image [43]. This makes it possible to create a custom mask with the exact size and shape of the space that is to be filled and use that to find the average colour instead of the predefined circle. This produces the average colour of every space in the image exactly at the expense of adding an extra stage of flood filling before the Canny output itself is filled (a stage of flood filling that would have to be undertaken on the rover in the full system). The consistency between images for this is the maximum possible based on colour alone (Figure 4.3), though inconsistency in the edge detection causes certain spaces to combine and divide constantly, leading to a small amount of colour inconsistency to remain regardless. This method has a noticeable impact on performance, though within acceptable bounds [DATA?], leading this to be the chosen method to be implemented into the full system. Examples of the results produced by the final single computer based build can be seen in Appendix D.

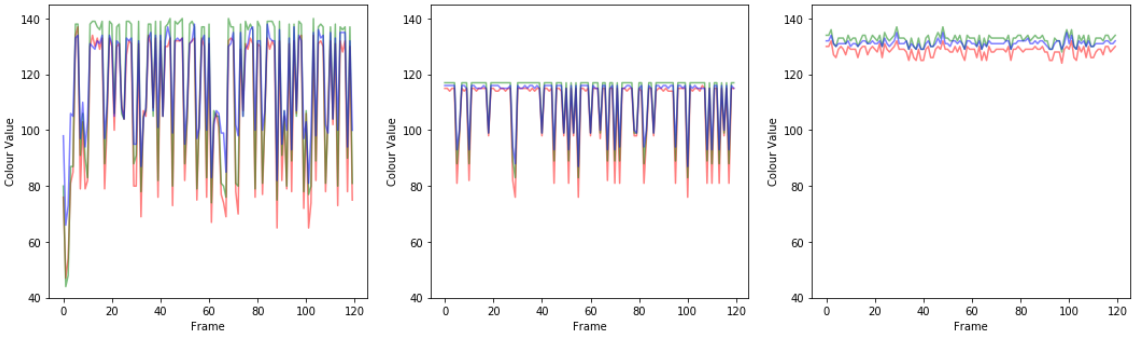


FIGURE 4.3: Comparison of Colour Averaging Methods. These are RGB values over time produced by flood filling an example area using the seed point colour (left), circle average (middle), and flood fill average (right). The increase in consistency from left to right is very apparent.

Chapter 5

Rover Implementation

The aim of this project is the development of a VR teleoperations system. This necessitates the procurement of some form of robotic platform to demonstrate the system on. This platform must be easily modifiable and cheap enough to be within budget, so it was determined that the most logical solution was to design a simple rover for the project; this is both cheap and allows for complete customisation with minimal hassle. The internals of the rover are shown in Figure 5.1, and a block diagram of the hardware in Figure 5.2.

FIGURE 5.1: Rover Internals. The left picture is a top-down view of rover, in which you can see the Pi on the right, the Il Matto mounted vertically at the top, the gimble at the bottom, and the stepper motor drivers for the gimble on the breadboard on the left. The right picture is of the inside of the rover, taken through a hatch in its underside. In this picture you can see velcro strips marking where the batteries are attached, a power distribution board and fuse in the centre, 2 DC motor drivers just below that, and 2 of the 4 DC motors in the top corners.

Due to the rover being a simple test platform for the proposed system, its hardware is mostly irrelevant to this report and therefore will not be discussed in detail (a detailed breakdown can be found in Appendix C). The application of computer vision techniques in an embedded system has high processing requirements, leading to the selection of a Raspberry Pi 3 as the core of the system (it was the highest performance embedded device readily available).

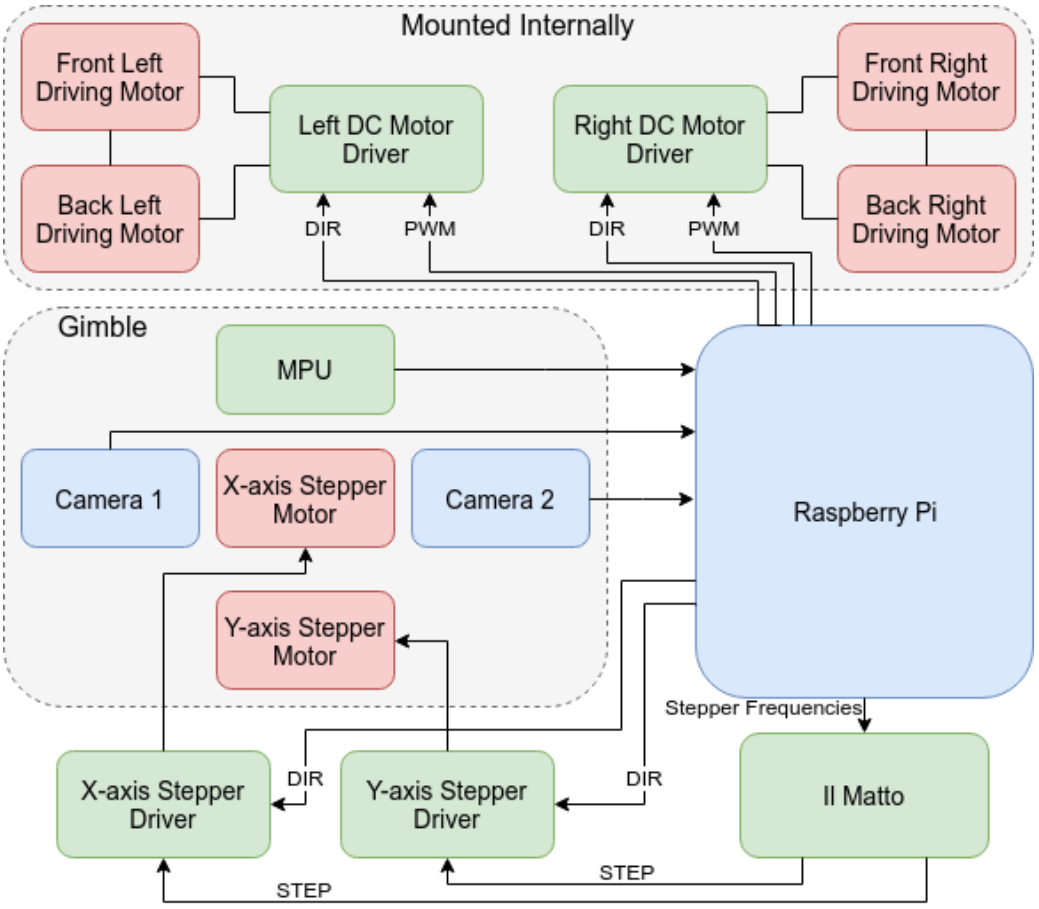


FIGURE 5.2: Hardware Block Diagram. The red blocks are motors, green blocks are control system components, and the blue blocks are part of the image pipeline.

5.1 Gimble Design

The choice of cameras had to fulfil a very specific set of requirements. The two cameras must be same, as any differences in the images due to the cameras would reduce the quality of depth map produced by the block matching algorithm on the server.

This makes the most obvious camera choice, the R-Pi camera module, unusable, as the Pi cannot use two simultaneously. The two cameras must also be able to take pictures on command from the Pi with low latency, reducing the possible options down to primarily USB webcams. Finally, they must have a high shutter speed. Any motion blur in the images will blur all the edges they contain, making them undetectable by the edge detection algorithm, and any morphing of the image while under motion due to the time it takes the shutter to pass across the entire sensor will once again reduce the quality of the depth map; a high shutter speed reduces motion blur and shutter related morphing, therefore making it essential for whenever the rover is in motion. This requirement reduces the possible cameras down to primarily dedicated computer vision cameras, however these are far outside the budget of a 3rd year project and are often too large to build a gimble for without also buying expensive motors.

[FPV CAMERA APPENDIX?]

Only one camera was found that fulfilled all these requirements while being cheap enough to fit within budget- the PlayStation 3 (PS3) Eye. The PS3 Eye is a camera for the PS3 to allow for games that incorporate aspects of computer vision, so it is designed with computer vision and value for money in mind. While the image quality is fairly poor, it is sufficient for the system to function reliably.

The design of the gimble (Figure 5.3) has considerable impact on the 3D environment the system produces. As mentioned in Section 2.6.3, the closer the cameras are to parallel with each other, the less the images have to be rectified before the depth map is generated. Similarly, the stability of the gimble is very important, as any vibrations will cause inconsistency in the alignment of the cameras, leading to inaccurate depth maps. This led to the chosen design where the X-axis motor is located centrally, between the cameras, to balance the weight around the rotational axis of the y-axis motor. The 3D-printed part the cameras are attached to is also stabilised through mountings on both sides of the x-axis motor, using a ball bearing on the side not driven by the motor.

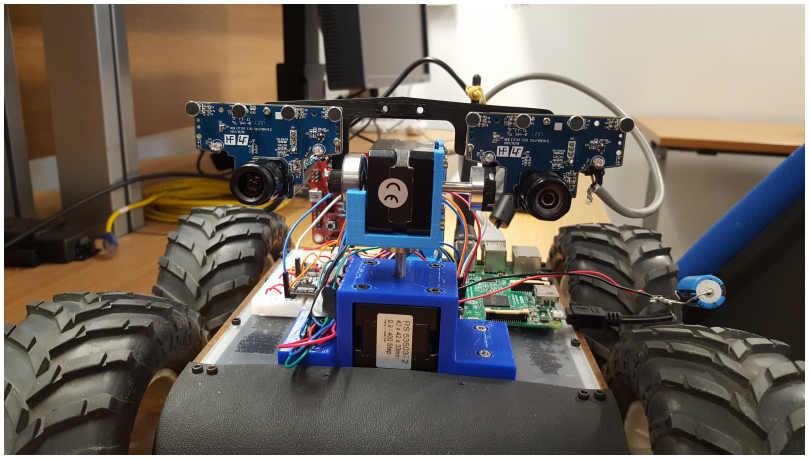


FIGURE 5.3: Gimble Picture. The Y-axis motor is housed at the bottom, with the X-axis motor directly above it. The ball bearing that stabilises the non-driven side of the cameras' backplate can be seen to the left of the X-axis motor.

Another important aspect of the gimble design is the distance between the camera lenses. The further apart the two cameras are, the closer distance objects will be in the depth map. Ideally we would want to match the interpupillary distance of human eyes (63mm on average [44]), so objects in the 3D environment appear as close as they would were the user standing in the place of the robot. However, with the X-axis motor located centrally, it is not possible to produce that distance between the camera lenses. The inter-lens distance in the final design is 120mm, as this is the closest distance possible without reducing the stability of the gimble. While not ideal, it simply results in objects appearing closer in the depth map than they actually are and a longer distance from the cameras where an object is too close for a distance to be calculated (the cameras are "cross-eyed" if you will).

5.2 Image Pipeline

As can be seen in Figure 3.2, the rover takes a picture with both cameras, abstracts those images, then sends that data off as a single combined packet to the server. While Chapter 4 discussed the abstraction process as a single step that produces a full abstraction with both edges and spaces filled with colour, this does not reflect the implementation utilized in the full system. As previously mentioned, the final

step of filling the spaces in the edge detected image using the selected seed points and average colours is done on the server. Also, only one of the two images needs colour information at all, as the edges are the only part of the abstraction required to produce the server depth maps; the colours are to be used as a texture on the 3D environment this produces, and therefore only one set is required. Therefore, the data packets being sent to the server are made up of 2 bitmaps of the edge detected images and a set of seed points with their corresponding colours for one of the images.

Attempts to implement this process on a single thread on the Pi, as tested successfully on a laptop, either crashed the Pi or would produce a frame rate of around 1fps; the Pi's resources are not even close to adequate. To combat this, both pipelining and parallelism were utilized to make better use of the Pi's quad core processor (Figure 5.4), and compromises were made in the quality of the abstractions to reduce the workload.

The capture thread simply handles signalling the cameras to take a picture each. The capture and decoding of the images are done as separate operations to make the capture operation shorter and therefore the two cameras capturing as close to simultaneously as possible with a single thread. The first compromise in quality in favour of performance is made here, where the images are captured with a resolution of 320x240 rather than the cameras' standard resolution of 640x480. As will be discussed in greater depth later in this section, the highest workload task in the pipeline is flood filling. It can therefore be inferred that reducing the pixel count of every space by a factor of four would have provide a significant improvement in performance. The smaller images also have the benefit of lower detail in high detail areas, so sections that would become areas of dense lines when edge detected (examples of this effect can be found in Appendix B) will be significantly less dense, containing less extraneous data.

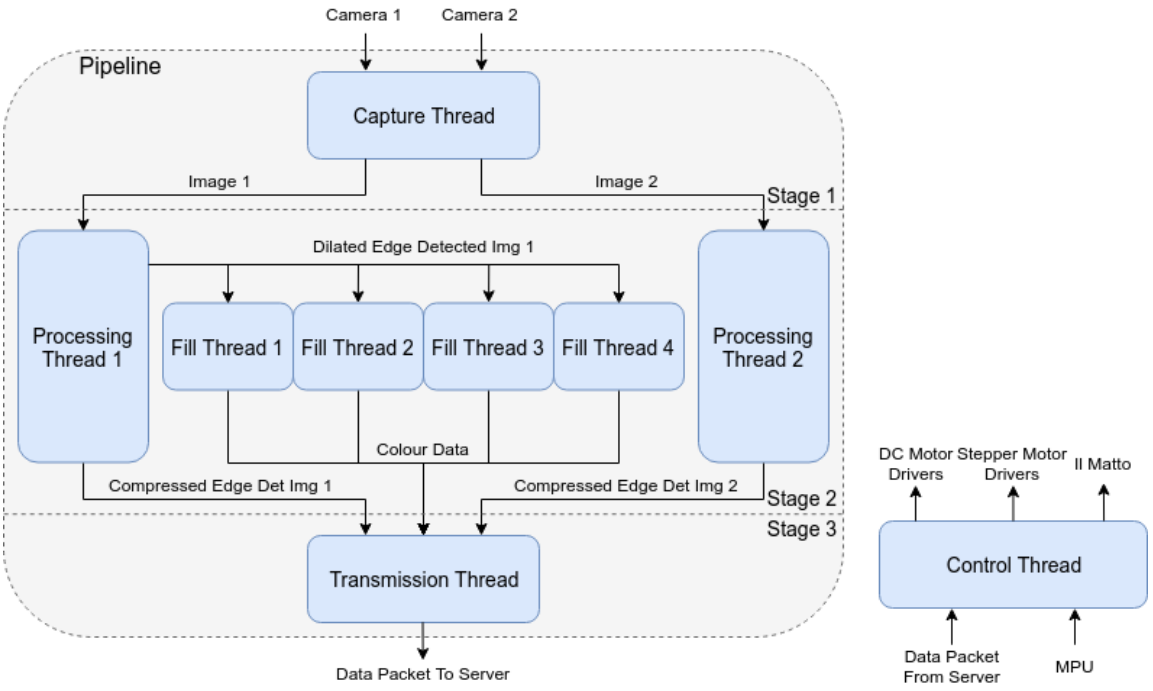


FIGURE 5.4: Raspberry Pi Threading Block Diagram.

The processing threads are concerned with the edge detection and compression of the images. The edge detection process is mostly as described in Chapter 4; the only difference is only the image being sent from processing thread 1 to the fill threads undergoes the final dilation step. The purpose of the dilation is to bridge gaps between the edges, creating defined shapes for the flood fill process. A side effect of dilation is a reduction in edge accuracy, which is very important when the images are to be used to produce depth maps later on. This leads to the logical conclusion that the images should be sent without dilation, and dilation only applied to find the colour data in the fill threads and then again on the server to create the complete coloured abstraction, allowing the depth maps to be constructed with non-dilated images.

While it can be induced from comparing the original images to their edge detected versions by eye that the latter contains less information than the former, this will only be reflected in real numbers if file format and compression are considered carefully. When the common image formats are compared, PNG would appear to be effective for this use case [45], as it excels at efficiently storing large blocks of the same colour (most of each edge detected image is black space). When tested on the edge detected

images (Figure [COMPRESSION COMPARISON]), it was confirmed that PNG provided the lowest file size, and compression level 5 provided the best ratio between file size and compression time (using the imencode function in OpenCV). More intelligent compression was tested using libimagequant [46], however it resulted in very poor performance (<2fps) with negligible improvement to the level of compression. An alternate option to compression as a bitmap was vectorising the images, however this was inaccurate and not as effective as bitmap compression (more information can be found in Appendix D).

As previously established, the stage in the image pipeline that has the highest performance impact is the colour averaging, due to its use of flood filling. As the number of spaces being flood filled is determined by the number of points from the sobol sequence being sampled, the number of sobol points is an important variable in tuning the performance of the system. In the testing done on a laptop in Chapter 4, the number of sobol points being used per image was always between 400 and 600, and this filled a reasonable number of spaces while providing a reasonable frame rate of [REDO FPS TESTS]. When this was attempted on the Pi however, the frame rate was significantly below 1fps. This called for a significant reduction in sobol points and the application of parallelism; good performance was achieved using 4 threads dedicated to the flood fill colour averaging task, each taking 15 sobol points. A total of 60 sobol points is a significant reduction from the laptop implementation of the algorithm, however it must also be noted that the images being used are much smaller, so do not require a large number of sobol points to have been mostly covered (Figure [ROVER/LAPTOP ABSTRACTION COMPARISON]). While it may seem as though the parallelization of this process could cause issues if two threads are attempting to fill the same space in the image simultaneously, this simply results in two successful seed points with slightly inaccurate average colours assigned to them, and only one of them being used to create the complete abstraction on the server. As the colours of the spaces are only recorded to provide the user a better understanding of the objects in the environment, it is not concerning if occasionally one is not an exact average.

Once the three pieces of data for an image pair (two compressed edge detected images and a set of colour data) have been generated, they must be combined into a single packet and sent to the server; this is covered by the transmission thread.

While the images are at this stage already compressed into a set of bytes that can be transmitted as is, the colour data needs its own custom packaging. Each seed point-average colour pair is formed into its own sub-packet with the structure shown in Table 5.1. These sub-packets make up the colour data data segment within the complete data packet.

TABLE 5.1: Colour Data Sub-Packet Structure. The seed point components are given 2 bytes each due to their maximum values being larger than a single byte can store.

Byte in Colour Data Sub-Packet	1	2	3	4	5	6	7
Usage	Seed Point				Average Colour		
Component	X		Y		Red	Green	Blue

As the size of each data segment is unknown and highly variable, knowing where one ends and another begins on the server is a challenge. The solution utilized in this instance is separating each data set with a splitter made up of 7 bytes that contain 0, 1, 2, 3, 4, 5, and 6. This sequence of bytes is extremely unlikely to occur within the data, so can be easily used to pinpoint the starts and ends of the data sets. This therefore leads to the data packet format shown in Table 5.2.

TABLE 5.2: Data Packet Format.

Data Segment	Colour Data			Splitter	Img 1	Splitter	Img 2
Contents	Sub-Packet 1	Sub-Packet 2	...	0...6	PNG Data	0...6	PNG Data

5.3 Control System

Xbox controls

Data packet format

PID loop

Chapter 6

Server-Side Implementation

The purpose of the server is to receive image data from the rover, produce a 3D environment from it, and feed back control inputs from the user. This is all done within the framework of the 3D game engine Unreal Engine 4 [47]. Unreal 4 was chosen because it provides easy interfacing with almost any VR headset on the market without large rewrites of the code, it is simple to integrate with OpenCV, and is free to use.

6.1 Depth Mapping

6.2 Coloured Abstraction Construction

As previously established, to produce the final texture that is applied to the 3D environment, a complete coloured abstraction must be produced from the dominant camera image and colour data.

6.3 3D Environment Generation

Chapter 7

Evaluation and Conclusion

Evaluation of fps and response time Evaluation of abstraction quality Evaluation of environment quality

Comparison of product and brief Evaluation of project management Conclusion

Bibliography

[1] K. M. Lee, "Presence, explicated," *Communication Theory*, vol. 14, no. 1, pp. 27–50, 2004.

[2] J. M. Loomis, "Presence in virtual reality and everyday life: Immersion within a world of representation," *PRESENCE: Teleoperators and Virtual Environments*, vol. 25, no. 2, pp. 169–174, 2016.

[3] S. A. McGlynn and W. A. Rogers, "Considerations for presence in teleoperation," in *Proceedings of the Companion of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, HRI '17, (New York, NY, USA), pp. 203–204, ACM, 2017.

[4] M. Bounds, B. Wilson, A. Tavakkoli, and D. Loffredo, "An integrated cyber-physical immersive virtual reality framework with applications to telerobotics," in *International Symposium on Visual Computing*, pp. 235–245, Springer, 2016.

[5] IrisVR, "The importance of frame rates," 2017. Available: https://help.irisvr.com/hc/en-us/articles/215884547-The-Importance-of-Frame-Rates. [Accessed: April 2018].

[6] M. Borg, S. S. Johansen, K. S. Krog, D. L. Thomsen, and M. Kraus, "Using a graphics turing test to evaluate the effect of frame rate and motion blur on telepresence of animated objects," in *GRAPP/IVAPP*, 2013.

[7] E. Ackerman, "Oculus rift-based system brings true immersion to telepresence robots," 2015. Available: http://spectrum.ieee.org/automaton/robotics/robotics-hardware/upenn-dora-platform. [Accessed: April 2018].

[8] M. Gupta, F. Hennecker, T. Isaacs, M. Sharhan, and L. Stauskis, "Biologically inspired robots: A vision system inspired by the human eye," 2016. Electronics and Computer Science, University of Southampton.

[9] Virtual Reality Society, "What is virtual reality?," 2017. Available: https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html. [Accessed: April 2018].

[10] Oculus, "Oculus rift homepage," 2017. Available: https://www.oculus.com/rift/. [Accessed: April 2018].

[11] HTC, "Htc vive homepage," 2017. Available: https://www.vive.com/uk/. [Accessed: April 2018].

[12] N. R. Council, *Virtual Reality: Scientific and Technological Challenges*. Washington, DC: The National Academies Press, 1995.

[13] T. Fong, "Interactive exploration robots: Human-robotic collaboration and interactions," 2017.

[14] NASA, "Robonaut 2 homepage," 2016. Available: https://robonaut.jsc.nasa.gov/R2/. [Accessed: April 2018].

[15] RCExplorer, "Fpv starting guide," 2009. Available: https://rcexplorer.se/educational/2009/09/fpv-starting-guide/. [Accessed: April 2018].

[16] N. Smolyanskiy and M. Gonzalez-Franco, "Stereoscopic first person view system for drone navigation," *Frontiers in Robotics and AI*, vol. 4, p. 11, 2017.

[17] J. I. Lipton, A. J. Fay, and D. Rus, "Baxter's homunculus: Virtual reality spaces for teleoperation in manufacturing," *IEEE Robotics and Automation Letters*, vol. 3, no. 1, pp. 179–186, 2018.

[18] P. Wang, J. Xiao, H. Lu, H. Zhang, R. Yan, and S. Hong, "A novel human-robot interaction system based on 3d mapping and virtual reality," in *Chinese Automation Congress (CAC), 2017*, pp. 5888–5894, IEEE, 2017.

[19] O. MAHDI, M. Alshujeary, and A. Jasim Mohamed, "Implementing a novel approach an convert audio compression to text coding via hybrid technique," 11 2013.

[20] A. Barrett-Sprot, "Data abstraction for low-bandwidth communication," 2017. BEng Project Report, Electronics and Computer Science, University of Southampton, Available: https://secure.ecs.soton.ac.uk/notes/comp3200/e_archive/COMP3200/1617/abs1g14/pdfs/Report.pdf. [Accessed: April 2018].

[21] J. Ribas-Corbera and J. Sklansky, "Interframe interpolation of cinematic sequences," *Journal of Visual Communication and image representation*, vol. 4, no. 4, pp. 392–406, 1993.

[22] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1874–1883, 2016.

[23] S. Joe and F. Y. Kuo, "Constructing sobol sequences with better two-dimensional projections," *SIAM Journal on Scientific Computing*, vol. 30, no. 5, pp. 2635–2654, 2008.

[24] L. Grünschloß, "gruenscholoss.org," 2017. Available: http://gruenschloss.org/. [Accessed: April 2018].

[25] Wikipedia, "Sobol sequence," 2017. Available: https://en.wikipedia.org/wiki/Sobol_sequence/. [Accessed: April 2018].

[26] BMVA.org, "What is computer vision?," 2017. Available: http://www.bmva.org/visionoverview. [Accessed: April 2018].

[27] D. H. Ballard and C. M. Brown, "Computer vision," *Prenice-Hall, Englewood Cliffs, NJ*, 1982.

[28] OpenCV team, "Opencv," 2017. Available: https://opencv.org/. [Accessed: April 2018].

[29] R. Jain, R. Kasturi, and B. G. Schunck, *Machine vision*, vol. 5. McGraw-Hill New York, 1995.

[30] R. Maini and H. Aggarwal, "Study and comparison of various image edge detection techniques," *International journal of image processing (IJIP)*, vol. 3, no. 1, pp. 1–11, 2009.

[31] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[32] A. Mordvintsev and K. Abid, "Opencv canny edge detection tutorial," 2013. Available: http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html. [Accessed: April 2018].

[33] V. Jaimini, "Hackerearth.com flood fill tutorial," 2017. Available: https://www.hackerearth.com/practice/algorithms/graphs/flood-fill-algorithm/tutorial/. [Accessed: April 2018].

[34] E.-M. Nosal, "Flood-fill algorithms used for passive acoustic detection and tracking," in *New Trends for Environmental Monitoring Using Passive Systems, 2008*, pp. 1–5, IEEE, 2008.

[35] N. Qian, "Binocular disparity and the perception of depth," *Neuron*, vol. 18, no. 3, pp. 359–368, 1997.

[36] G. Linda and C. G. Shapiro, *Stockman, Computer vision*. Prentice Hall, Upper Saddle River, NJ, 2001.

[37] opencv dev team, "Camera calibration," 2014. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_calib3d/py_calibration/py_calibration.html#calibration. [Accessed: April 2018].

[38] S. Ghost, "Stereo calibration using c++ and opencv," 2016. Available: http://sourishghosh.com/2016/stereo-calibration-cpp-opencv/. [Accessed: April 2018].

[39] J. J. L.Alvarez, R.Deriche, "Dense disparity map estimation respecting image discontinuities," 2000. Available: http://serdis.dis.ulpgc.es/~lalvarez/research/demos/StereoFlow/index.html. [Accessed: April 2018].

[40] K. R. Fall and W. R. Stevens, *TCP/IP illustrated, volume 1: The protocols*. addison-Wesley, 2011.

[41] J. Postel, "User datagram protocol," tech. rep., 1980.

[42] OpenCV Documentation, "Opencv: ffilldemo.cpp," 2017. Available: https://docs.opencv.org/3.3.0/d5/d26/ffilldemo_8cpp-example.html. [Accessed: April 2018].

[43] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library.* " O'Reilly Media, Inc.", 2008.

[44] N. A. Dodgson, "Variation and extrema of human interpupillary distance," in *Stereoscopic Displays and Virtual Reality Systems XI*, vol. 5291, pp. 36–47, International Society for Optics and Photonics, 2004.

[45] P. Aguilera, "Comparison of different image compression formats," *ECE*, vol. 533, 2006.

[46] K. Lesiski, "libimagequant-Image Quantization Library," 2017. Available: https://pngquant.org/lib/. [Accessed: April 2018].

[47] Epic Games, "What is unreal engine 4?," 2018. Available: https://www.unrealengine.com/en-US/what-is-unreal-engine-4. [Accessed: April 2018].

Appendix A

Contour Based Seed Point Location

The ideal place to aim to flood fill a space from would be the centre point of the space. OpenCV provides the functionality to take the Canny output image (a matrix of colour values) and convert it into a set of contours. Contours are line objects stored in a hierarchical structure and have functions that can provide the centre point of each contour. Although the centre points of the contours will not map exactly to the centre points of the space, they are close enough approximations to flood fill from (Figure A.1).

Unfortunately, due to a combination of the processing time required to convert the lines into contours and the number of contours produced that have no impact on the spaces left to be flood filled, this method is too resource heavy to produce 10 fps on a laptop, therefore is also too resource heavy for use on the raspberry pi.

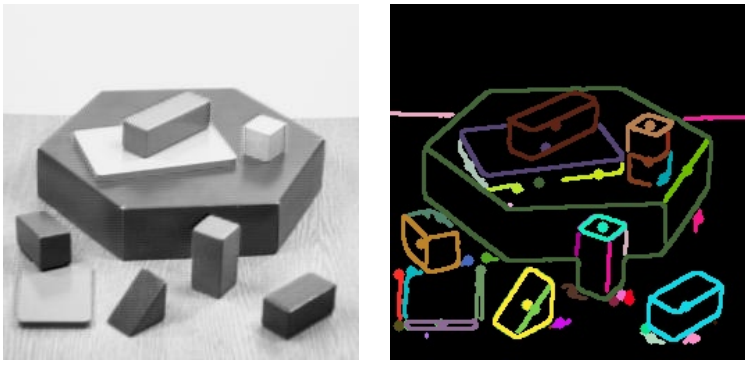


FIGURE A.1: Demonstration of contour centre point location. The original image (provided by OpenCV) on the left has been Canny edge detected, these edges have converted into contours, and the centre points of these contours located. The results of this are displayed on the right, with each contour and its corresponding centre point in a different colour. It can be observed that the centre points provide adequate coverage of the black spaces in the image to be used as seed points for flood filling.

Appendix B

Demonstrations of Full Data Abstraction

FIGURE B.1: Demonstration of full abstraction process. The final parameters and methods (of those under discussion) are a blur kernel of 5x5, a low threshold of 25 (for this example), Sobol seed point generation, and averaging via preliminary flood fill. It can be seen that most areas of the image are being effectively edge detected and flood filled with the correct colours; however, some areas are hard to interpret such as around the bushes in the bottom left, and some spaces have been left unfilled such as the panel below the top left corner of the building. These issues are minimal though, therefore leading me to conclude that the process is effective at producing recognisable abstractions of the input images.

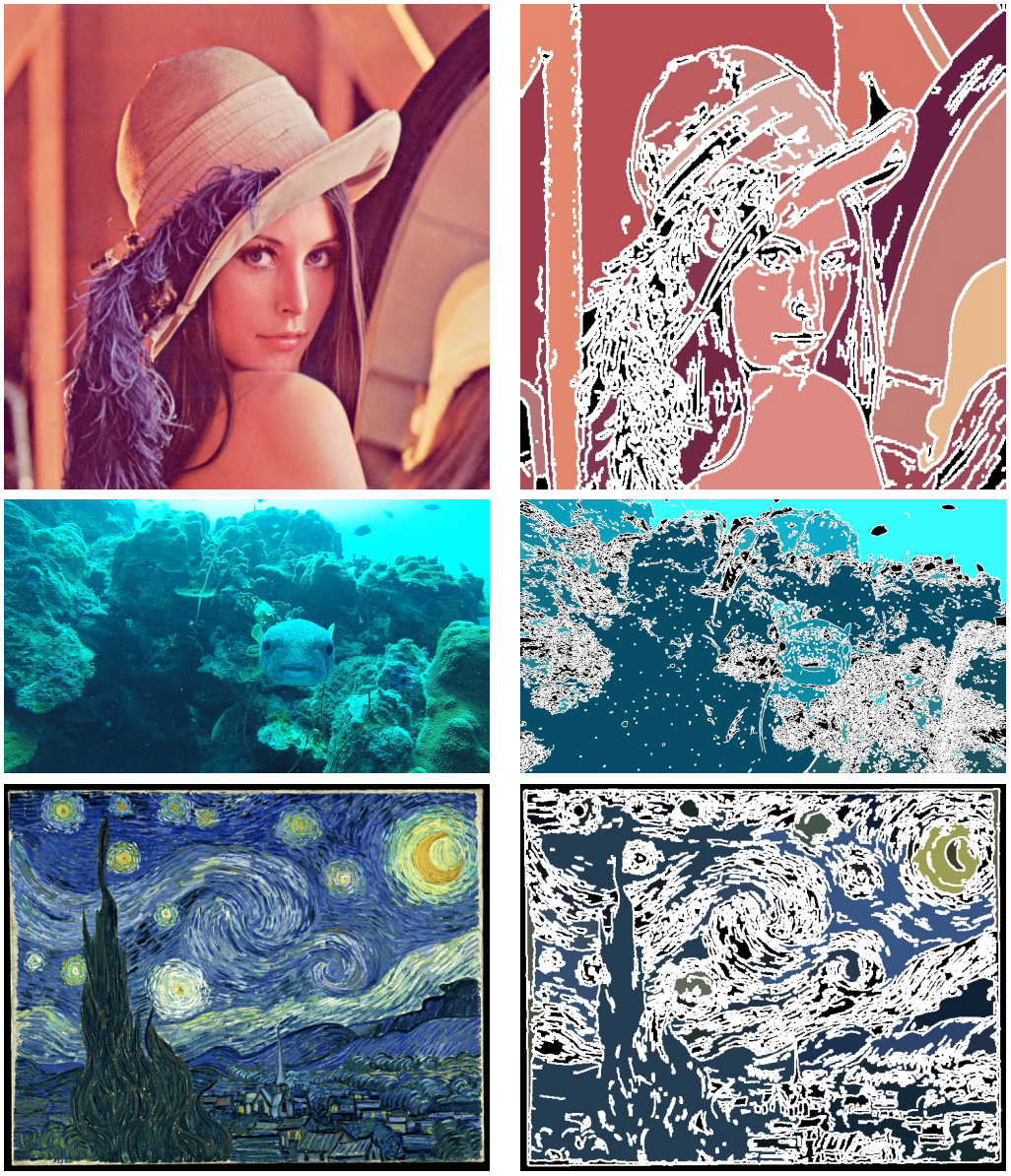


FIGURE B.2: Additional examples of the full abstraction process. The top image (provided by OpenCV) is very simple, so the abstraction gives clean and easily recognisable results. The middle image (provided by fellow student Tom Darlison) was selected due to its limited colour palette and poor focus to test the limits of the process. While the image has mostly filled to a single colour, the pufferfish, rocks, coral, background fish, and open water are all identifiable. The bottom image (provided by OpenCV) was selected due to the extreme number of edges, due to the very clear brush strokes. Once the Canny threshold was turned down considerably, the result was recognisable as Starry Night, though much of the image has been overwhelmed by the edge detection lines. It can be concluded from these tests that the data abstraction process is effective at producing recognisable images, however the difficulty in interpreting the abstracted images is heavily dependant on the focus and complexity of the image.

Appendix C

Rover Hardware Breakdown

Appendix D

Vectorization

Appendix E

Project Brief

Using Data Abstraction and Inter-Frame Interpolation for Low Data Rate Communication Between a 3D Camera and VR Headset

Adam Melvin am20g15@soton.ac.uk
Klaus-Peter Zauner kpz@soton.ac.uk

There are many challenges, such as monitoring hostile environments, that call for the use of remotely controlled robots. Often in these scenarios, it would be useful to be able to view the environment with a sense of depth to better understand the scale, and dangers, of the robots surroundings. This can be provided through the use of a 3D camera on the robot and Virtual Reality (VR) goggles, however due to the minimum frame rate that can be displayed in a VR headset without causing motion sickness in the user being 60fps (optimally 90fps is preferable), a comfortable and useful experience would often require an unfeasibly high wireless data rate.

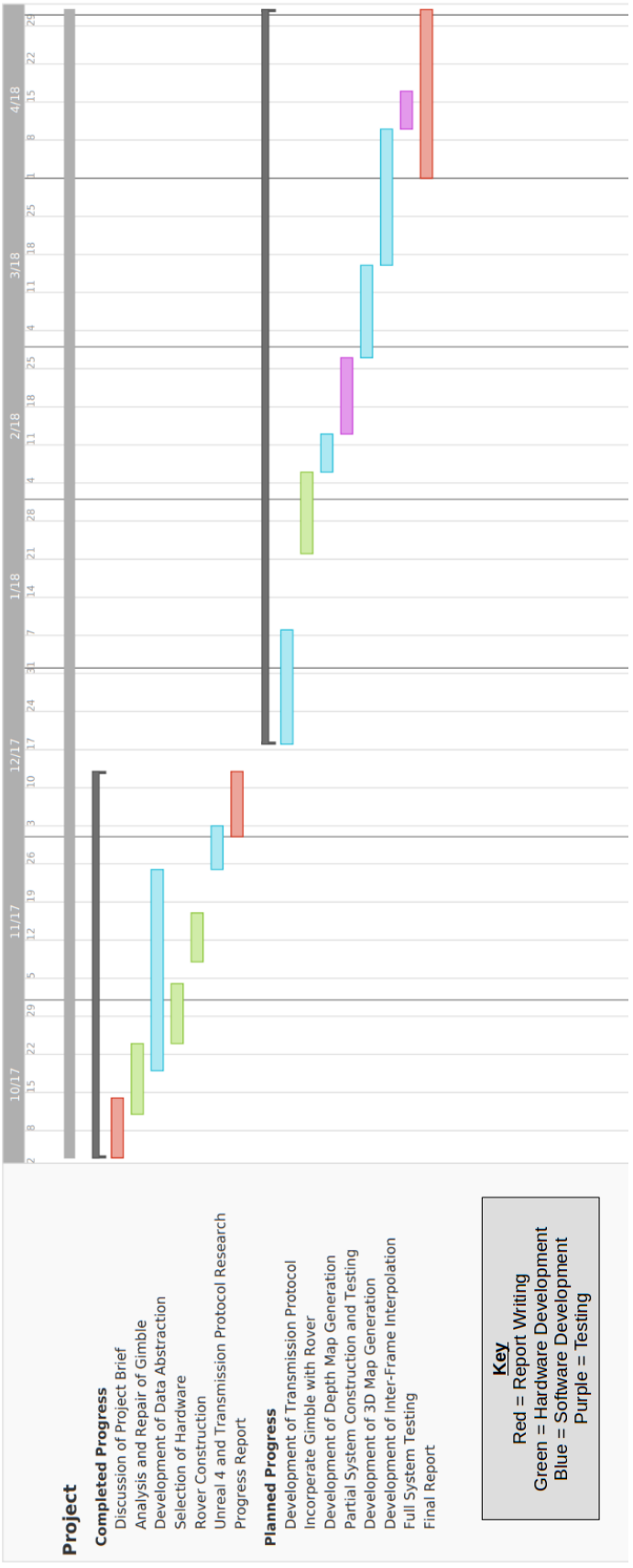
The aim of this project is to significantly reduce the data rate required to be transmitted between a robot and teleoperator through the use of data abstraction and inter-frame interpolation. A 3D camera rig mounted on a remotely controlled rover is to take around 10 pictures per second, then they are reduced down to the minimum amount of data required to identify the objects in the environment. These much smaller images are sent wirelessly to a server, where they are used to create a 3D map of the environment. These 3D maps are analysed and intermediate frames are estimated to increase the frame rate from 10fps to 60-90fps. This much higher frame

rate estimated map of the environment can then be displayed in the VR headset through the use of a video game engine. Although the estimated frames will not accurately represent the real world, the comfort they provide the user will allow them to focus on the transmitted information without feeling ill.

A simple rover and off-the-shelf VR equipment will be used as a foundation for the system. Different camera systems and interpolation algorithms will be tested on this foundation to discern the setup that produces the best ratio between data rate and usability.

Appendix F

Gantt Charts


Project
Completed Progress
Discussion of Project Brief
Analysis and Repair of Gimble
Development of Data Abstraction
Selection of Hardware
Rover Construction
Unreal 4 and Transmission Protocol Research
Progress Report
Planned Progress
Development of Transmission Protocol
Incorperate Gimble with Rover
Development of Depth Map Generation
Partial System Construction and Testing
Development of 3D Map Generation
Development of Inter-Frame Interpolation
Full System Testing
Final Report
Key
Red = Report Writing
Green = Hardware Development
Blue = Software Development
Purple = Testing
10/17
11/17
12/17
1/18
2/18
3/18
4/18

Appendix G

Costing and Risk Assessment

Component	Supplier	Price	Quantity	Delivery
Edimax USB 2.0 Wireless Adapter	RS	£17.05	1	£0.00
Nylon XT60 Connectors Male/Female (5 pairs)	Hobby King	£3.00	1	£4.66
XT60 Harness for 2 Packs in Parallel	Hobby King	£1.94	1	
GEP-XT60 PDB BEC 5V & 12V	Hobby King	£3.04	1	
Turnigy 2650mAh 4S 20C Lipo Pack	Hobby King	£12.28	2	
MAX14870 Single DC Motor Driver Carrier	HobbyTronics	£5.40	2	£2.88
HS1177 Sony Super HAD II CCD FPV Camera	Unmanned Tech	£21.00	2	£3.50
Black EasyCAP USB Video Adapter	GearBest	£11.29	2	£0.00

Figure G.1: Hardware Costs List. This shows the use of budget for this project thus far. Total budget spent = £124.97

Risk Event	Likelihood (1-5)	Impact (1-5)	Risk Exposure (1-25)	Action
Unable to complete system in time	4	4	16	System is built from the input to the output, allowing for the submission of the working section instead
Response time of the system is unacceptable for use in VR	3	5	15	System is designed and being build with runtime performance as the highest priority
Inter-frame interpolation does not perform as intended	3	3	9	System can function without this component
Data abstraction process is not accurate enough to form a depth map from	2	4	8	Areas where refinement is possible are known so the process can be made more accurate if neccessary
EasyCap devices are never available or do not work with the Raspberry Pi	4	2	8	Preparations have been made to use my supervisor's budget to replace FPV cameras with webcams
Data abstraction process is too resource intensive for the Raspberry Pi	2	3	6	Areas where quality reductions could be made in exchange for performance are known
LiPo battery combusts	1	4	4	LiPo batteries are supervised while charging, charged in a flame-retardant bag, and only connected in parallel when the same voltage
Hardware failure	1	2	2	Remaining budget can be used to replace failed component

FIGURE G.2: Risk Assessment for the project going forward.