University of Southern Queensland
Faculty of Engineering & Surveying

A.R Drone Vision-Guided Searching

A dissertation submitted by

Derek Long

in fulfilment of the requirements of

ENG4112 Research Project

towards the degree of

Bachelor of Engineering(Mechatronics)

Submitted: October, 2013

Abstract

UAVs are becoming more and more capable of performing complex tasks in today's world. Military and scientific applications are becoming increasingly less dependant on human monitoring, with the increased capabilities of UAV sensors and control abilities. There is also an increasing need for automation of repetitive tasks in industries around the world. There is amazing opportunity for process efficiency improvement if a machine can be substituted in place of a person to perform a menial tasks.

This dissertation is aimed at getting a UAV to perform an automated function for use in monitoring or search and rescue. It proposes a method for an A.R. Drone to home in on a object "beacon" in a set search area. Having UAVs automated in this way has the potential to both improve monitoring process that already exist, and open up new opportunities for aerial applications that were previously not possible.

The original implementation of this homing application achieved the base function only. The drone was able to search a small area for a coloured ball, and navigate to it, once the ball had been located. Several tests were performed in preparation for this, developing machine vision and tracking functions that would be needed for the drone to perform such a task.

Work at this stage is being done to further investigate how to enhance the abilities and efficiency of the algorithms that form the foundation of the application. Detection robustness and control smoothing are priorities in improvement, as well as developing additional features, such as obstacle avoidance.

University of Southern Queensland
Faculty of Engineering and Surveying

ENG4111/2 *Research Project*

Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

Prof F Bullen
Dean
Faculty of Engineering and Surveying

Certification

I certify that the ideas, designs, and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Derek Long

0050109483

24/10/2013

Acknowledgments

Firstly, to my parents, for supporting me to get through university and making every effort to help me start my career.

To my two supervisors, Dr. Tobias Low and Dr. Cheryl McCarthy, for always being available to give advice/feedback, and for giving valuable input into the write-up.

And finally, to the awesome people with whom I spent time with as a student over the 4 years at USQ. Without your company, I would have found it hard to enjoy the university experience, and to give my full effort into study.

Derek Long

University of Southern Queensland

October 2013

Contents

List of Figures

Listings

Chapter 1

Introduction

This dissertation aims to show development of the capabilities of a flight drone to perform tasks (detailed in the dissertation objectives) automatically using its sensors and hardware alone, instead of relying on direct user control. This will be accomplished by application of the areas of machine vision and machine automation, working in unison.

1.1 Objectives

There are two specific objectives for this project. They are:

1. To get an flying drone to perform a simple homing task. This homing task will be generically defined as searching for an object in the surrounding area, and navigating to it once it has been found.

2. To investigate machine vision methods or processes that will allow the drone to perform this homing task more efficiently or under difficult conditions. Possible conditions that will be used in this project as are increasing the complexity of the object being used as a beacon for the homing task, or adding environmental

challenges such as uneven elevation, inconvenient light conditions, or obstacles that need to be navigated around or through.

1.2 Project Background

1.2.1 Machine Automation

In the recent history of mankind, machine automation has had a huge impact on the economic, scientific, and military structures around the world. The ability to give a machine instructions, and sensor input has removed many, sometimes tedious jobs that people once had to do themselves. Another use for machines that has become possible with developing technology is the ability to make them go places that people cannot, often for scientific purposes such as exploration. Machines are continuing to become 'smarter' in the present day, with developing technology making them more and more capable of acting in the place of people.

Manufacturing has been one of the sectors most greatly affected by this technology. As machines became able to do more complex tasks on their own, without direct human control, the amount of product that companies have been able to produce increased at amazing rates. Less workers were needed in the manufacturing process, and whilst jobs were lost this way, jobs were also created in the maintenance of these machines. Resource gathering capability has increased in line with manufacturing, with more methods available and more assistance provided in the processes by machinery.

Science has also been aided by robots and other advanced equipment. In a similar fashion to automated manufacturing, scientists have set up repetitive scientific or medical testing to be performed by computers and machines. Robots have been used to explore and perform testing in locations or conditions that are beyond the ability of humans to endure. Examples of locations are underwater or space environments. Example of conditions that humans could not survive are environments of extreme heat

or radiation.

This area of developing technology has also found military use. The capability of using a robot (a UAV in some cases) to perform a reconnaissance or combat role instead of putting human lives in danger has proved invaluable to many nations.

1.2.2 Machine Vision

Input from sensors is vital to machine automation. Actions have been based on the input from sensors measuring weight, heat, and all kinds of physical properties. An image from a camera is another kind of input. Computers can dissect these images, and use them to provide information to a user or a program.

Machine vision is the field dealing with computerized processing and interpretation of images. Successful implementation of machine vision allows a processing device to use input from a camera to detect objects, or otherwise gather information in order to 'see and act' automatically in the place of a person. It is yet another aspect of machine automation, that is heavily being researched and developed today. There are always more improvement to be made, more functions to give a computer, in order to allow it to make decision or execute pre-programmed responses based in its visual input. Computerized image processing has become more powerful as computational power available in computers has increased. Today, personal computers are able to quickly perform image processing on a photo, or even be able to work at sufficient speed as to keep up with a live camera feed input, upwards of 20-30 frames per second, at remarkable quality. This processing power has certainly not been publicly available for too long. Only machines on the mainframe or supercomputer level would have been capable of this 10-20 years ago.

The applications of machine vision are as broad as the application of machine automation. Manufacturing can be optimized with machine vision allowing controlling computers to visually assess the state of the product (detecting faults, blemishes, etc.).

Some of the scientific or military applications discussed earlier would be impossible without the assistance of machine vision.

1.2.3 Flight Drone Automation

Automation in aerial vehicles has allowed the vehicle to fly without a pilot. These Unmanned Aerial Vehicles (UAVs) have already been used in scientific and military applications. The scientific uses of UAVs will be what this project will attempt to look into and develop. Machine vision assists UAVs in gathering information concerning points of interest on the ground, and in navigating along its flight path. The later use for machine vision will be focused on in this dissertation.

As stated earlier, this project aims to use machine vision and robot automation together to get an AR Drone to perform vision dependant task automatically, instead of being directly controlled by a user. The main objective is to develop the drone's ability to navigate automatically based on its visual input.

There are a lot of functions that can be looked into once effective machine vision based control is implemented on the drone. Some more advanced functions could be obstacle detection and avoidance, or performing much more difficult homing tasks, such as finding a man in a field. Application of UAVs with powerful machine vision may include assisting in search and rescue, or being better able to explore areas unreachable by man.

The image processing techniques need to be developed to make these things possible. As mentioned, these techniques are ever being improved, to make machines closer able to mimic people when making decisions based on visual input. These techniques would then be able to control the AR drone in a similar manner to the image processing methods performed in this project.

1.3 Background of Project Resources

From the start of the project, there were two resources that were defined as ideal for the first platform in testing. These were a specific quadcopter and a machine vision library (to be implemented on it), respectively.

The backgrouds of these two resources will also be covered in the introduction to this project.

1.3.1 A.R. Drone 2.0

In 2012, Parrot released its A.R. Drone 2.0 for purchase (pictured in Figure 1.1). It is designed to be a recreational quadcopter, but has very impressive hardware and functionality for its price.

Figure 1.1: A.R. Drone 2.0

Specifications are as listed directly from Parrot (Parrot, 2013)[1].

[1]The only hardware differences between the A.R. Drone 1.0 and the 2.0 were that the 1.0 had no

- Ultrasound sensors for altitude measurement.
- 3 axis gyroscope, accelerometer, and magnetometer.
- High definition camera (1280x720) on the front of the drone.
- Standard definition camera (640x480) on the belly of the drone.
- Indoor and outdoor swappable shells.
- USB port for additional modules.
- WiFi connection to controlling device.
- Smartphone, tablet, or computer control.
- Approximate weight of 400 grams.
- Signal range of 50 meters.
- Estimated battery life of 10 minutes.
- Priced between $300 to $400 AUS.

The drone has a interfaces very well with its controlling device and operates smoothly. The drone takes off and lands smoothly, and auto-stabilizes very well in mid air. There is an emergency stop available to the user, as well as emergency protocols on the drone (if one of the blades contacts an object, for example). Users are able to switch between camera feeds, and are able to save images at any instant from either camera.

There are a lot of extra features added to the AR Drone to make it appealing to buyers. One of these is a GPS flight recorder that connects to the open USB socket, sitting above the battery. With this module, users will be able to record entire flights as video files to the recorder. They will also be able to utilize the GPS function to plan flight paths and monitor the drone's progress through a flight.

magnetometer, and the front facing camera was not high definition.

Image processing capabilities have also been officially released. The drone is able to recognize certain objects, identifying unique colours patterns on objects[2]. This allows drones to see each other, and Parrot have utilized this ability to create some fun applications, like an app that allows two people to simulate a dogfight with their AR Drones.

Other items identifiable by colour include a cap that users can wear, allowing the drone to know the location of the pilot, and the inner ring on a race track 'checkpoint' prop that users can set up and use to simulate races on another official Parrot application.

A Software Development Kit (SDK) has been released, giving developers the opportunity to create their own applications for the AR Drone. It has been released on the iOS, Android, and Linux platforms, offering great variety for developers. Example applications with source code released in this SDK include a video stream example, which looks exclusively at the process of receiving and displaying the camera feed, and a full application, which runs with all drone systems enabled and displays input from every sensor. These examples will be very useful in learning how the drone works and how to add to the current image processing sequence.

1.3.2 OpenCV

OpenCV (standing for Open Source Computer Vision) is a function library that focuses on real-time machine vision and image processing. It has been released to interface with C/C++, Java, and Python[3], offering a great variety of platforms available to the user. OpenCV has a sizeable community contributing and developing for it, and promises to continue to be a useful tool for image processing tasks well into the future.

OpenCV includes a lot of general necessary functions for working with images, such a saving, loading, and displaying images. There is also a huge range of colour-space

[2]The detection functions are accessible, and may be used in testing.

[3]The two platforms available for use in this particular project will be C as used in the Linux examples and Java for the Android phones.

conversion functions available. The rest of the OpenCV library is mainly dedicated to image manipulation and processing techniques. Some of the techniques that are well suited to achieving the machine vision implementation that we are seeking in this project will be outlined and discussed in the later in this report, as we review some OpenCV learning books.

Chapter 2

Literature Review

The two primary areas where previous research can be drawn on to assist this project are the processes involved in the image processing, and the structure behind the drone control. The review of literature for this project will follow the general areas of research in chronological order of encounter. Other optional areas related to this task will also be reviewed.

2.1 UAV Image Processing Hardware

One of the first specifications that needs to be made when working with a UAV in research is how to process the data. This is mainly a question of "on-board" processing as opposed to remote processing through means of wireless communication, both with advantages and disadvantages.

One of the primary reasons to use remote processing is control. A fitting example of the accessibility remote processing gives research is the work done on UAV Identificiation (Junkui, Zaikang, Fugui & Tao 2010). The researchers were able to easily monitor flight data through the laptop, in addition to sending flight instructions a wireless signal in the case of manual control. This setup greatly assisted the team in getting the project

to a state where the UAV could perform its task without supervision. Although not done here, the team could also have used the laptop to process data for the UAV.

The end goal of automation tasks, such as these being done with UAVs, is to have them operating completely independently. To achieve this, the UAV needs be capable of processing all incoming data itself, instead of sending it away to be handled by a base station. Most research is directly on UAVs with on-board processing (Phang, Ong, Yeo, Chen & Lee 2010). This ensures that the UAV can react to changes as fast as possible, and keeps the supporting hardware requirements of the UAV to a minimum.

For this project, remote processing is going to be necessary, as it brings benefits to the experimentation process as well as satisfying the processing needs that the A.R. Drone has. Both manual control and data monitoring will be necessary in the early stages of testing in this project. Manual shutdowns will be written into the program, and all flight data will both be displayed and logged on the base station.

2.2 Mobile Robot Sensor Configurations

The available sensors for image capture will also affect the capabilities of the drone. In this case, as per the re-defined resources, we have a drone with a single camera on the front, and a single camera on the bottom.

Several papers have shown that the ability for a robot to detect a known object is strengthened with other configurations. A computer receiving both colour images and depth images has much more information to process, but will yield a much higher accuracy in detection (Baum, Faion & Hanebeck 2012). Another configuration is stereo vision, in which two cameras show views from different angles, allowing three dimensional information to be constructed of the scene (Khateeb, Awang & Khalifa 2009).

The A.R. Drone that will be used in this project, however, has only a single camera in each facing (front and downward). So the techniques shown above may not be able to

be directly applied. The multi-sensor setups may be more viable on a different drone.

2.3 Processing Efficiency

There is a limit to the processing power at our disposal, and so the efficiency to the processing methods are of concern. The requirements of the chosen method will affect what platforms are able to be used in the final implementation. Currently, the two options are an Android smart phone, or a laptop computer.

Methods that reduce processing requirement whilst still achieving the same result will be needed to bring the functions and capabilities of this application to work on a smart phone. One method that will be investigated is using scanlines in object tracking as proposed by (Park & Lee 1995).

The use of scanlines will reduce the amount of information to be processed by the controlling device by limiting this stream of information to only a few lines and columns of the image, instead of the entire thing. Several different methods have been shown in the paper, for different shapes to be tracked. Of most interest out of these shapes, at least initially, is the circle. The demonstrated algorithm could be applied to take the application from Linux and test it on an Android phone.

One of the prerequisites of this algorithm is that it is dependent on knowing the starting position of the object and the distance from the camera to the object. We will not have these two things when searching an area for the ball, as the position of the ball is originally unknown. This means that the scanline algorithm cannot be implemented until the object is first detected.

This should not prove to be a huge disadvantage when implementing the scanline algorithm. One thing that can be told to the program is the size of the object. We will therefore know how big the object will appear at any distance, and we can check the object when it is first discovered. The distance the object it from the camera can be

therefore estimated, in sufficient accuracy. The scanline method can then be used to track the position and distance of the object, as the drone navigates to it.

Processing efficiency improvement has also been demonstrated in several other papers (Chung & Yang 1995) (Hong, Chun, Lee & Hong 1997), by reducing image size being processed, once a target area within the full image has been identified as a Region of Interest (ROI). Whilst processing efficiency was a much more general problem back in the time these papers was published, the findings are still relevant today, especially given that the ideal platform for this project's implementation has severely less available processing power than a full computer.

2.4 Object Detection Methods

The task of the image processing is to identify the target object within an image. A few of the common methods used in research will be discussed.

2.4.1 Colour Searching

The simplest method of detecting an object can be used if the object has a known colour. For example, by searching an image for "skin" colour, people can be easily detected in an image (Feyrer & Zell 1999).

The process works by converting an RGB image to the HSV (Hue, Saturation, and Value) colour space, to separate the colour value of each pixel from the saturation and brightness values. Given this separation, a simple search can be performed on each pixel, showing whether it lies within the desired colour range or not.

Tutorials showing OpenCV implementations of this method are available on the internet. So, if given an object that has a unique colour, this method will be the first one to try.

2.4.2 Histogram Backprojection

histogram projection algorithms have also been used in object identification. The method has been applied in the development of a grey-fuzzy controller (GFC) for motion control of a tracker (Luo & Chen 2000).

Histogram Projection makes use of the colour information in an image, which is usually more reliable than monochrome information. The colour model of an RBG histogram can be made from an image, the algorithm then matches this histogram with a reference histogram of the target object in a database, and generates a backprojected image. The object can be detected by further processing of this image. The algorithm is fast, but subject to limitations in illumination variance.

This method may be one of the best options when looking to detect and navigate to ore complex objects. In the initial stages, an object that is easily detected (a bright and stand-out colour) has been used, and advanced methods have not been needed to identify it. However, this algorithm will allow more complex objects to be identified. The colour of the object will not be the focus anymore, but the pattern of colours that makes up the perception of the object. Implementations of this algorithm can be found in various OpenCV documents and books. The effectiveness of this method will be reported in the implementation section of this dissertation.

2.4.3 Classifiers

Another method for detecting an object is use of a classifier algorithm. A classifier uses feature detection algorithms to analyse an image, and relies on some user input for statistical training in detecting specific objects. This is achieved by feeding several images into the algorithm that contains the object, so it can learn what to look for. After it has been trained, it can find these objects in other images on its own.

Classifier algorthims have previously been tested on drones for various tasks (Majidi

& Bab-Hadiashar 2005), as well as on other mobile systems (Kim, Lee, Young-Chul, Kwon & Park 2011).

This technique will be the last to try in this project, due to the complexity specification (found in the project specification). If time permits, this method will be used to try and more reliably detect complex objects in the later stages of this project.

2.5 Object Tracking

The issue of properly tracking a moving object has been around for many years. Many different techniques have been developed to address this need. Several methods focus on interpreting the contours (formed by colour variation) of an image (Allili & Ziou 2006) (Seo, Choi & Lee 2004). Other methods rely on the matching of feature points between frames (Zheng, Xu, Dai & Lu 2012), or mathematically predicting the path through forming models for various objects (Luo & Chen 2000). There are several models that have been developed for specific objects, such as the Gaining Angle of Gaze (GAG) model that has been implemented in ball-catching tasks (Mori & Miyazaki 2002).

A specific application to UAV tracking of target objects have been recently tested (Teuliere, Eck & Marchand 2011). The UAV was used to track a toy car (shown in Figure 2.1).

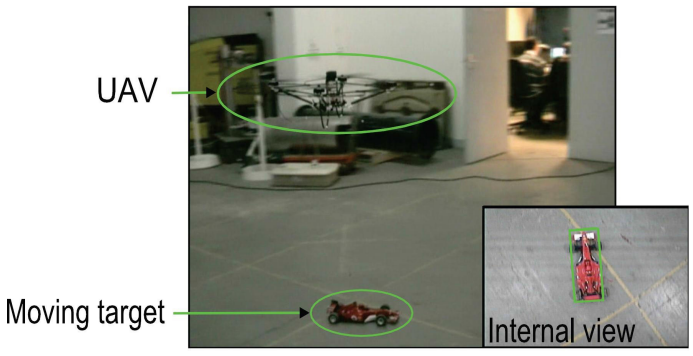


Figure 2.1: Setup for the Car Tracking Experiment

Some of these techniques also utilize motion detection by image subtraction (subtracting the previous frame's pixel value from the current frame's value) to assist the process.

The specified conditions for the testing will eliminate the need for advanced tracking techniques, in that the object being targeted is stationary, and any movement in the video feed will be due to the movement of the drone. Because it is entirely the platform that is moving, we will have all the information necessary to predict the movement of the object relative to the camera, should we need it.

2.6 Drone Control

The next problem that we are presented with after looking at the image processing side of things is the control of the drone. The control commands will need to be controlled by the result of the image processing.

The combination of image processing and drone control has been achieved (Teuliere et al. 2011). They successfully got a drone to hover over a target and pursue it when it moved. The feedback scheme of the drone in the forward direction was a combination of position feedback and velocity feedback. The position feedback gave a distance between the drone and its target position, which drove it, and the velocity feedback acted as damping. The yaw was also controlled to match the orientation of the target.

A similar system will be useable in this project, though the drone will be getting the information a different way. Instead of using the bottom camera the whole time, the front camera will be for first to detect the target, providing an approximate distance, if the objects size is known. The drone can use this information to head towards the object, switching to the vertical camera when it is close enough. The drone can then centre its position over the target.

2.6.1 Direct Path Pursuit

Another method of pursuit was demonstrated, which involved a robot following the observed path of the target, rather than pursuing it directly (Feyrer & Zell 1999). Memory capability was also present in the car-tracking experiment shown above. If line of sight to the toy car was lost (Figure 2.2 shows a chair obstructing vision), the UAV would follow the expected path of the car, based on its last known heading and speed. The UAV would then re-establish vision of the car on the other side of the obstacle.

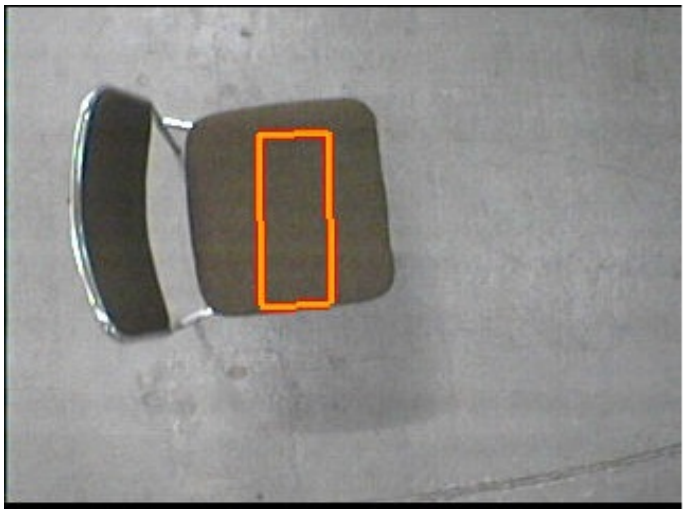


Figure 2.2: Car Tracking Obstacles

Later in the project, more advanced pursuit techniques will be considered to be if they will improve the drone's performance in the task. Memory functions may be used to assist in object tracking, particularly to stabilize uncertain detection methods, such as the *cvHoughCircles* function (used to find a hoop in the later experiments. Tests will be done to test the default implementation of the algorithm, as well as any modifications necessary.

2.7 Obstacle Avoidance

Another possible path of expansion of the drone capabilities is obstacle avoidance capability. Work on integrating this ability into a fixed-wing UAV has been done (Lee, Lim & Kim 2011). The developed model allows the UAV to remain free and out of the way of both stationary and moving obstacles, whilst maintaining line of sight to its target.

Although this has been implemented on a different form of UAV, the theory behind it is no less relevant. Once again, this is being considered optional, but will be a high priority for expanded capability in the homing task.

Obstacle avoidance will have a much simpler application in this project. There will not be need for much calculation on object paths and trajectories, as they are not moving. The drone can simply navigate around/through the obstacle first and then focus back on the beacon object.

2.8 Drone Path Planning in Search and Rescue Operations

The path that the drone will fly while it is searching for the beacon object is also something that needs to be considered.

Algorithm development and testing and has been shown. The algorithm constructs a search path based on the probability distribution (of finding the object) over the area (Lin & Goodrich 2009).

Other work in the area of path planning involves the focus on planning a search path around multiple objectives (Jilkov, Li & DelBalzo 2007), and in planning optimized paths for multiple UAV use (Bellingham, Tillerson, Alighanbari & How 2002) (Rasche, Stern, Kleinjohann & Kleinjohann 2011).

Implementing search path planning may come in later in the project, but is definitely being considered as optional. In the original conditions, we will assume that there is an equal chance of finding the object over the entire area, with no section having a higher probability than any other. The drone will simply be pro-programmed to search either in a circle around itself, or in a cone in front of the take-off position. This will be done to allow focus on the image processing methods and drone control integration, but path planning will be implemented as well, if there is sufficient time.

In the case of search a circular area, the drone will simply turn on the spot until the object is found. This is still satisfying the requirement of finding the object before setting out to reach them, but without the effort in programming a search path into the drone. Another way of performing a quick and easy search may involve using case code to step the drone through pre-programmed movements. This isn't putting path planning algorithms to use, but may be sufficient to perform testing.

Chapter 3

Methodology

This section contains all of the considerations that go into a properly executed research project. This is made up of the project methodology, risk assessment, project content licensing and project timeline [1]. Future implications of the results of this dissertation will also be considered.

3.1 Project Methodology

As with any research task, this project will need a structured approach to research and testing in order to successfully achieve its desired goals as stated in the project objectives. As this project involves repetitive experimentation with changing conditions, a basic method for performing an individual experiment must be formed.

The current version of this method is as follows:

1. Set conditions for the experiment. This includes specifying the object that will be used as a beacon, setting environment condition such as lighting or elevation (if any), and specifying obstacles that will be used in the environment.

[1] Whilst the project timeline is also part of this section, it can be found instead in Appendix B.

2. Identify the specific needs in image processing techniques and drone control that will be needed to satisfy the set conditions.

3. Research image processing techniques from sources including OpenCV to develop a machine vision solution adequate for the desired conditions.

4. Test image processing solution on the video stream example provided in the AR Drone SDK.

5. Repeat steps 3 and 4 until a satisfactory solution is achieved.

6. Research drone control techniques and develop method for the machine vision to guide the drone to the beacon.

7. Test full solution in an application.

8. Repeat steps 6 and 7 until a satisfactory solution is reached.

Certain steps are expected to take much longer to complete on the first iteration of these experiments. Steps 6 and 7 are an example; they will take much longer when first investigating how to manually pre-program a search path into the drone, and how to establish a relationship between machine vision and drone navigation. Once these principles are grounded in the first experiment, performing them in subsequent iterations should be a much easier task.

3.1.1 Project Development

The first iteration of the experimental process (image processing implementation) will be under the simplest conditions possible, to allow possible speed bumps in the project (such as learning to control the drone by feedback from the camera) to be passed as quickly as possible. Once the initial test has been completed, the rest of the project time will be spent performing as many iterations of the experiment as possible, each time with hardened conditions.

The original testing conditions are defined below.

- No challenges in the testing environment. No elevation differences, no obstacles, and no lighting inconsistencies will be allowed in the environment.
- Beacon object easily detected. A yellow ball will be used, making it detectable by colour.

The planned (as of this time) challenges to be added in subsequent experiments are in this order:

1. Placing an obstacle in the testing area. This obstacles could be a barrier or pole that needs to be avoided, or an object like a window or large tunnel that needs to be navigated through.
2. Removing the colour uniqueness of the object being used as a beacon. This will make it harder to detect. New image processing methods will be required.
3. Adding elevation of the beacon as a factor in the search (for example, making the beacons height higher than the hover height of the drone).
4. Adding lighting inconsistencies, to make the image processing more of a challenge.

In each case, OpenCV will need to be implemented in a new way to deal with the differing circumstances.

3.2 Risk Assessment

This section looks at the risks that will arise in the course of this project, and the measures that can be taken to minimize them.

3.2.1 Identification

The primary risk when performing tests in this project will be contact between the drone and the surrounding environment, potentially cause damage to both. Several safety measures will have to be enforced to preserve the condition of the drone and the people and environment that it will be put near, over the course of this year.

3.2.2 Evaluation

Testing of an object detection algorithm carries a high likelihood of errors or accidents. As the code is in development, the drone will frequently do what it isn't supposed to. Whilst measures can be taken to reduce the likelihood (see Risk Control), they cannot be eliminated.

The hazard found in this project testing is the set of rotating blades on the drone. The damage that they can do to a person is limited to fragile areas (eyes, etc). They cause nothing but a sting on contact with the skin. The damage is more likely to occur to the blades themselves or the surrounding environment. Both of these are rather minor, as the blades can be easily replaced, and the environment can be easily chosen to minimize damageable surfaces/objects.

3.2.3 Control

Several considerations will need to be made regarding the testing site. The first will be environment. A location with a large, flat area with no obstacles (such as building supports or walls) would be necessary to provide an ideal location for the original test, which will be under the easiest conditions (no elevation challenges, no obstacles, etc.). An well-lit indoor location would be ideal in terms of lighting, although a large outdoor field would also be adequate.

Supports and walls may be permissible in later stages of testing, when obstacles are being used. Caution will need to be taken, however, to ensure that the drone will not fly into these obstacles. Specialized testing of obstacle avoidance and navigating functions may be necessary, if such a testing location was to be used.

A couple of simple safety measures will be sufficient to conduct tests in an acceptable manner. These are:

1. The tester shall set up the controlling device a minimum of 3 meters from any point on the expected flight path of the drone.
2. The tester will remain within reach of the emergency stop for the duration of a test.
3. The tester will ensure that the area is closed off to the public before conducting tests.

As the drone's flight behaviour will not be determined by user input, special care will need to be taken to ensure that the drone will not navigate into obstacle (mentioned above), and will also keep within normal tilt limit when being controlled by programmed instructions instead of the normal coding. The manual emergency shutdown will need to be re-implemented on the tested applications, as well as making sure that the drone's automatic emergency protocols are still operating, for conditions such as breaching the operating orientation limits, or detecting obstruction on the blades.

In addition, the automated drone control should be set to navigate slowly. This may slow down testing a bit, but will be much safer by giving the user plenty of time to react and activate the emergency stop if the drone is heading towards an obstacle.

3.2.4 Project Consequential Effects

After the completion of this project, the programs may either be used, or further improved upon by other people in research. In both cases, responsibility will be on me to ensure that I am not liable from any damage caused in future use of this research.

The primary means by which this can be ensured is by provision of thorough documentation for aiding any user or developer in the future. The capabilities and limits/weaknesses of the developed code must be clearly stated - perhaps the easiest of future accidents might be a user misinterpreting these specifications. Research at such a level in the given timeframe will not produce a polished product, and that's what makes the fine details of the product so important.

If proper information is supplied in the dissertation, consequential effects may be minimized, and the research will be performed in an ethical manner.

3.3 Project Intellectual Property

The work done in this project, and the code that has been created will form a good starting point for any student who is seeking to go further into application development for the drone. Tasks that use the homing function as a base will be greatly benefited, in that the functional extension will be able to be fully targeted in research, rather than dealing with the homing ability, first.

To support the future work of any students working in the area, there will only be a request for attribution in the use or modification of the work presented in this dissertation. This means that only recognition is requested to build on the work presented here.

3.4 Experiment Methodology

This section will outline the requires in documenting the test runs over the course of this project.

Each test will be placed as a section, grouped into chapters by test type (visual recognition tests, feedback control tests, and homing tests). All code listings will be provided in an attachment to the submission (either by attachment to the back of the submission, or as included files).

The following list will detail (in order) the documentation requirements for each test.

- Objectives - the aims and goals of the test will be stated.
- Code Configuration - specific functions used in the test will be listed. A full flowchart of the image processing pipeline will also be provided.
- Test Setup - all assumptions and conditions set for the test will be listed, and pictures will be provided of the testing setup.
- Expected Results - the expected results given the testing conditions will be proposed, for later reference.
- Results - the results of the test will be documented. Both screenshots from the drone camera and logged flight data may be presented to show the drone's behaviour.
- Discussion - the results will be compared to the expectations, and the positives and negative sides of the results will be discussed.

Chapter 4

Program Functional Development

This chapter will look at the underlying initial stages that transform sample code available from the makers of the A.R. Drone into a functional base code for further development. This code must be capable of both displaying the image feed from the drone cameras (and be capable of performing image processing on them) and sending basic user commands to the drone (such as taking off).

The starting point that will be used is the `video_demo` example given in the A.R. Drone SDK. This code shows how to display the video feed from the drone's front camera. OpenCV has not been used to do this.

4.1 OpenCV Implementation in Video Feed

For testing of the first version of the image processing implementation, the video only sample code will be modified and used (`video_demo`). This will be done on the Linux platform, to make sure that the image processing will be able to keep up with the frame rate of the camera.

There is a tutorial posted online (Kout 2012) that shows how to modify the default

video processing stream in this example, and replace it using OpenCV. This allows the OpenCV library to be available to use in processing the images and turning them into input for the drone control thread.

4.1.1 Installation

Firstly, the required OpenCV packages needs to be installed. The needed packages are:

- libopencv-core
- libopencv-highgui
- libcv

All of the dev packages for the above installations should be installed, as well[1].

4.1.2 Includes

Two files need to be modified to include the OpenCV libraries in the build. The first is the make file. The 'pkg-config –cflags opencv' phrase needs to be added to line 45 to include OpenCV in the compilation process.

The same phrase needs to be added to line 48 in order to include OpenCV in the linking process. Both of these additions can be seen in most of the makefile listings given in appendix code listings.

The additional includes in the C file (`display_stage.c`) will just include the libraries `include <opencv/cv.h>` and `include <opencv2/highui/highgui.h>` for use[2].

[1] This is for the AR Drone SDK on Linux.

[2] I had to replace ¡opencv/highgui.h¿ with ¡opencv2/highui/highgui.h¿, to specify the exact location where my laptop installed the library.

4.1.3 Code Modification

The first source code modification necessary is to change the output format from RGB 565 to RGB 24. This brakes the sample method of displaying a frame, but allows OpenCV to be used. This is changed in `ardrone_testing_tools.c`, and can once again be seen in the code listing.

Secondly, a function to store the incoming data in an RGB image object is needed. This function will be called by the event function to store the image before displaying it[3].

Listing 4.1: Iplimage Function

```
IplImage *ipl_image_from_data(uint8_t* data, int reduced_image,
 int width, int height)
{
  IplImage *currframe;
  IplImage *dst;

  currframe = cvCreateImage(cvSize(width,height),
  IPL_DEPTH_8U, 3);

  dst = cvCreateImage(cvSize(width,height),
  IPL_DEPTH_8U, 3);

  currframe->imageData = data;
  cvCvtColor(currframe, dst, CV_BGR2RGB);
  cvReleaseImage(&currframe);

  return dst;
}
```

Thirdly, the event function needs to be modified to use the OpenCV method instead of the stock method. The commented out code has been removed to make reading easier:

Listing 4.2: `C_RESULT` Function.

```
C_RESULT display_stage_transform (display_stage_cfg_t *cfg,
vp_api_io_data_t *in, vp_api_io_data_t *out)
{
    uint32_t width = 0, height = 0;
```

[3]OpenCV uses BGR by default, and converts it to RGB. We may be able to skip this process in future work by converting to our target colour space directly from BGR.

```
    getPicSizeFromBufferSize (in->size, &width, &height);

    IplImage *img = ipl_image_from_data((uint8_t*)
        in->buffers[0], 1, 640, 360);
    cvNamedWindow("video", CV_WINDOW_AUTOSIZE);
    cvShowImage("video", img);
    cvWaitKey(1);
    cvReleaseImage(&img);

return C_OK;
}
```

With all of this code implemented, the resulting `video_demo` will not appear to run any differently right now, but with OpenCV available for processing, we are ready to move forward and implement our machine vision algorithm.

4.2 Object Colour Tracking

There is an online tutorial (Fernando 2012) shows how to search an image for a uniquely coloured object, and to track the position of that object. This will prove to be very useful, as one of the easiest object to search for is one of a unique colour.

Only elements of this example useful for the application will be discussed here, the rest will be omitted.

4.2.1 Binary Image

The first section of this tutorial shows how to obtain a binary image using a specific colour as a criteria. The step by step process is as follows:

1. Convert the RGB (or BGR, in this case) to HSV. This is a colour space that separates the colour component from the brightness and saturation, allowing colour criteria to easily be applied.

2. An empty binary image is created.

3. The binary image is then filled with white or black pixels, as the HSV image is compared to the set criteria. The criteria are specific value ranges for the hue, saturation, and value (HSV), representing the specific colour we are looking for[4].

4. The binary image is displayed.

The code in Listing 4.3 and the images taken from the drone in Figures 4.1 and 4.2 show this process.

Listing 4.3: Modified HSV Search.

```
//Modified code inside Iplimage function
IplImage* imgHSV = cvCreateImage(cvGetSize(currframe),
 IPL_DEPTH_8U, 3);
  //Change the color format from BGR to HSV
cvCvtColor(currframe, imgHSV, CV_BGR2HSV);
IplImage* dst = GetThresholdedImage(imgHSV);

//Threshold function
IplImage* GetThresholdedImage(IplImage* imgHSV){
      IplImage* imgThresh=cvCreateImage(cvGetSize(imgHSV),
          IPL_DEPTH_8U, 1);

      cvInRangeS(imgHSV, cvScalar(170,160,60),
          cvScalar(180,256,256), imgThresh);
return imgThresh;
}
```

As an option, the image may be smoothed using Gaussian kernal both before the conversion to HSV and after the binary image is obtained, using the *CvSmooth* command.

4.2.2 Binary Image Tracking

In order to get some information on the object out of the binary image, we need to use the function *cvMoments* to find the average x and y position of the searched pixels.

[4]In the example, they set the hue criteria to be 170-180, representing red. The saturation and value criteria are set to be quite wide, to accommodate for various lighting conditions.

Figure 4.1: Ball Search Colour Image

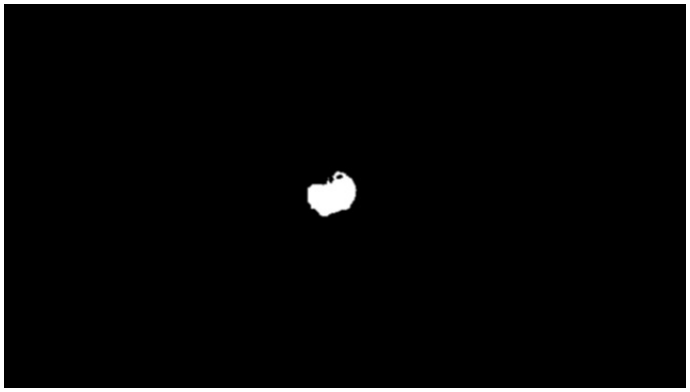


Figure 4.2: Ball Search Binary Image

Listing 4.4: Tracking Function.

```
void trackObject(IplImage* dst){
    // Calculate the moments of 'imgThresh'
    CvMoments *moments = (CvMoments*)malloc
        (sizeof(CvMoments));
    cvMoments(dst, moments, 1);
    double moment10 = cvGetSpatialMoment(moments, 1, 0);
    double moment01 = cvGetSpatialMoment(moments, 0, 1);
    double area = cvGetCentralMoment(moments, 0, 0);

     // if the area<1000, I consider that the there
         // are no object in the image and it's because
         // of the noise, the area is not zero
    if(area>1000){
        // calculate the position of the ball
        int posX = moment10/area;
        int posY = moment01/area;
```

```
            lastX = posX;
            lastY = posY;
        }

        free(moments);
}
```

With this tracking function in place, two integers (X and Y, for example) will store the position of the beacon object as seen by the camera. These variables can then be used to send flight commands to the drone. They can also be displayed on screen with the command *cvPutText*, or written to file using the standard file I/O libraries, in order to view and interpret the flight performance.

4.3 Additonal Features

All other additional code functions will be recorded here.

4.3.1 Region of Interest (ROI) Application in OpenCV

Research in OpenCV Implementation on mobile robots has already been performed (Deepthi & Sankaraiah 2011). The code is nearly directly applicable in this project.

One of the most useful code listings in this paper is the Region of Interest (ROI) separation from an image. It is designed to eliminate much of the image that doesn't need to be processed, in the interest of processing efficiency (as discussed above). The listing is shown in Listing 4.5.

Listing 4.5: Region of Interest Sample Code.

```
#Set Region of interest from the original image...
#Here cropping the image from minimum row index to
the max row of a cartoon strip
cvSetImageROI( original_image, cvRect(
min_col,min_row, (max_col-min_col),(max_rowmin_
row) ) );
#Create a cropped image from the source image
```

```
IplImage* cropped = cvCreateImage(
cvGetSize(original_image),original_image-
>depth,original_image->nChannels );
#Do the copy
cvCopy( original_image, cropped, NULL);
cvResetImageROI( original_image );
#Show cropped image
cvNamedWindow( "cropped",
CV_WINDOW_AUTOSIZE) ;
cvShowImage( "cropped", cropped );
cvWaitKey();
#Release the image window
cvReleaseImage(&cropped);
```

In the paper, the ROI was programmed to be centred on the area of the image that contained the most edges (as revealed from the edge-finder section of the image processing). For this project, the ROI could be defined as the section of the image containing the known object "beacon" (once it has been detected). This will allow the processing to be much faster once the drone has a general idea where to look for the object.

In this application, an ROI can be positioned around the target objects once they are found, massively reducing the processing needed in tracking the detected objects.

4.3.2 Circle Detection

Avoidance of a hoop is going to be worked into the later experiments, so a method of finding a hoop will be needed.

The most suitable function to use is the *cvHoughCircle* algorithm (from the OpenCV library). The function will detect circular objects in an image, and return the pixel coordinates (x, y, and radius) to an array. A sample test of this hoop is shown below, with code in Listing 4.6 and implementation in Figure 4.3.

Listing 4.6: Circle Detection Method

```
CvMemStorage* storage = cvCreateMemStorage(0);
  CvSeq* results = cvHoughCircles(
              imghoop,
              storage,
```

```
                CV_HOUGH_GRADIENT,
                4,
                200,500,200,50,400);

  for(i = 0; i < results->total; i++ )
{
    float* p = (float*) cvGetSeqElem( results, i );
    CvPoint pt = cvPoint( cvRound(p[0]), cvRound(p[1]));
    cvCircle(currframe,
                pt,
                cvRound( p[2] ),
                CV_RGB(0xff,0xff,0xff),1,8,0);
}
```

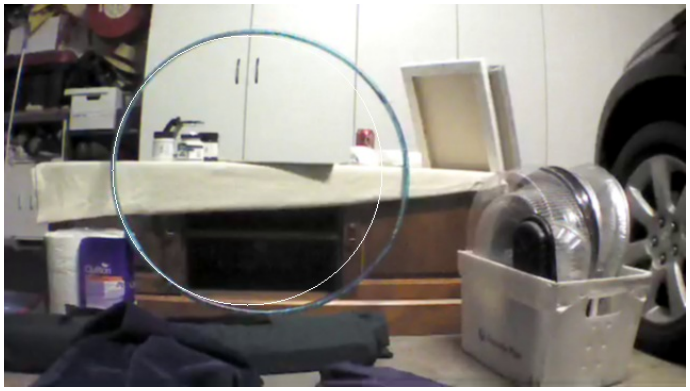


Figure 4.3: Finding a Hoop with *cvHoughCircle*

There are two primary improvement that will need to be made to the code in order for the machine vision to be reliable enough for use in any program. In its current state, the *cvHoughCircle* function has to be set to be so picky in order to avoid false detection, that even the target hoop will not be detected in most frames. In addition, the radius reading can be extremely inconsistent with each frame.

These two issues are addressed in Listing 4.7. The detection issue has been addressed by searching the image for the hoop colour first (shown in Figure 4.4), and then using the resulting binary image as input for the *cvHoughCircle* function. The radius issue has been addressed by adding some code in that filters out readings that are significantly different than the last.

Listing 4.7: Improved Circle Detection Method

```
imghoop = GetThresholdedImage(imgHSV);
```

```
  CvMemStorage* storage = cvCreateMemStorage(0);
  CvSeq* results = cvHoughCircles(
              imghoop,
              storage,
              CV_HOUGH_GRADIENT,
              4,
              400,300,100,5,400);

  // Updating Circle timer
  cir_t = t = ardrone_timer_elapsed_ms(&cir);

  if (results-> total > 0) {

  for(i = 0; i < results->total; i++ )
{
    float* p = (float*) cvGetSeqElem( results, i );

        // Only allow reasonable new readings through
    if (p[2] < (3*prev_radius/4)) {
    p[2] = prev_radius;
    }

    CvPoint pt = cvPoint( cvRound( p[0] ), cvRound( p[1] ) );
    cvCircle(currframe,
              pt,
              cvRound( p[2] ),
              CV_RGB(0xff,0xff,0xff),1,8,0);
    xposc = p[0] - (width/2);
    yposc = p[1] - (height/2);
    r = p[2];

    sprintf( buf, "%.0f %.0f %.3f", xposc, yposc, r);
    cvPutText(currframe,buf,cvPoint(50,50), &font1,
        CV_RGB(255,255,255));
    det = 1;
    started = 1;
    ardrone_timer_reset(&cir);
}

// In the case of a weird reading, use the previous one
} else if ((cir_t < 1000) && (det == 1)) {
  float xtemp = prev_x;
  float ytemp = prev_y;
  float rtemp = prev_radius;
  CvPoint pt = cvPoint( cvRound( xtemp ), cvRound( ytemp ) );
  cvCircle(currframe,
```

```
                pt,
                cvRound( rtemp ),
                CV_RGB(0xff,0xff,0xff),1,8,0);
    xposc = xtemp - (width/2);
    yposc = ytemp - (height/2);
    r = rtemp;

    sprintf( buf, "%.0f %.0f %.3f", xposc, yposc, r);
    cvPutText(currframe,buf,cvPoint(50,50), &font1,
        CV_RGB(255,255,255));

    ardrone_timer_update(&cir);
// what to do if it doesn't find a circle
} else {
  xposc = 0;
  yposc = 0;
  r = 0;
  det = 0;
  ardrone_timer_reset(&cir);
}


  // File writing function.
  if (started == 1) {
  ardrone_timer_update(&timr);
  t = ardrone_timer_elapsed_ms(&timr);
  }

  if ((started == 1) && (t < 10000)){
  oFile = fopen ("SRT_output.txt", "a");
  fprintf( oFile, "%.0f %.0f %0.3f \n", xposc, yposc, r );
  fclose(oFile);
  }

  prev_radius = r;
  prev_x = xposc + (width/2);
  prev_y = yposc + (height/2);
  cvReleaseImage(&imghoop);
  cvReleaseImage(&imgHSV);
  return currframe;
}

IplImage* GetThresholdedImage(IplImage* imgHSV){
        IplImage* imgThresh=cvCreateImage(
            cvGetSize(imgHSV),IPL_DEPTH_8U, 1);
        cvInRangeS(imgHSV, cvScalar(15,30,30,0),
```

```
            cvScalar(30,256,256,0), imgThresh);
        return imgThresh;
}
```

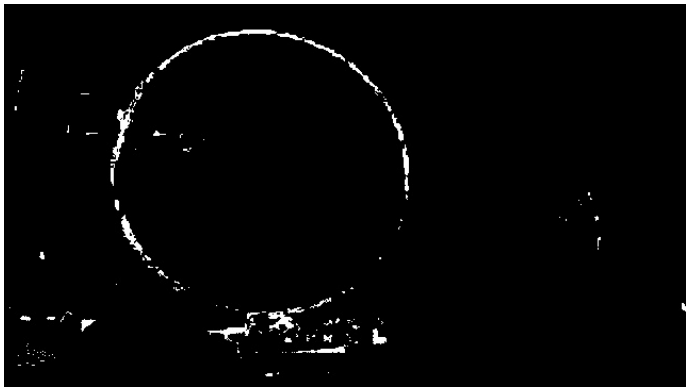


Figure 4.4: Hoop Binary Image

4.3.3 Histogram Backprojection

OpenCV has functions for calculating an image histogram and backprojecting it on an image. This can be implemented in the drone program by taking a picture of our target object, storing the image histogram, and backprojecting the histogram against each incoming frame, in search of the target.

Listing 4.8 shows an attempted implementation, placed in the image processing stage of the program.

Listing 4.8: Histogram Backprojection Implementation

```
imgref = cvLoadImage("../sample1.jpg", CV_LOAD_IMAGE_COLOR);
cvCvtColor (imgref, imgref, CV_BGR2HSV);
hue = cvCreateImage( cvGetSize(imgref), IPL_DEPTH_8U, 1);
int ch[] = { 0, 0 };
cvMixChannels( &imgref, 1, &hue, 1, ch, 1 );

backhist = cvCreateImage(cvGetSize(img), IPL_DEPTH_8U, 1);
int numColorBins = 256;
float range[] = {0, 255};
float *ranges[] = { range };
CvHistogram *hist = cvCreateHist(1, &numColorBins, CV_HIST_ARRAY,
ranges, 1);
```

```
// Calculate Histogram
cvCalcHist(&hue, hist , 0, 0);
```

Although repeated debugging and testing was performed with a sample object, the function (as of the end of this project) is still not operational. The crashing line is the last of the above code, *cvCalcHist*, used to point the object histogram at the incoming frame.

Part of the reason that there was so much trouble getting this to work was the limited support for C. Most OpenCV tutorials and discussion are all on C++, which has a differently defined function for *cvCalcHist*. This code will still serve as a good starting point (for performing backprojection) for anyone else who may work with the drone in the future.

4.3.4 Thread Management

Thread management will be necessary in running this application. It is uncessessary for the entire process of frame capture, processing, and resultant action to be put in the one thread, with the commencement of each action dependent on the completion of the last.

In order to completely separate these three processes, a thread would need to be developed for each. Each thread would only work with the latest information provided to it by the previous thread. This would be an efficient way to run the application, but there are a few drawbacks. The biggest drawback in this instance is that images would need to be sent between threads. This is a lot more memory consuming than simply sending numbers or coordinates between threads, which would be more preferable.

To organize the threads with this in mind, the final division (as will be used in all testing) is between the image capture and processing, and the drone control. The control thread will constantly sending the latest available command to the drone, and the other thread is dedicated to constantly updating this value for the needed movement.

The code for the control thread is shown below in Listing 4.9. This thread will be consistent in all of the experiments where the drone is flying. The thread will be absent from experiments in which the drone doesn't fly.

Listing 4.9: Control Thread

```
DEFINE_THREAD_ROUTINE( control_system , data )
{
    int seq_no = 0,prev_start = 0;
    control_data_t r_control , prev_control;

    PRINT( "Initilizing Thread 1\n" );
    while(r_control.start == 0)
    {
    vp_os_delay(10);
        //Taking Control Mutex
    vp_os_mutex_lock( &control_data_lock );
        r_control.start = control_data.start;
    vp_os_mutex_unlock( &control_data_lock );
    }

    ardrone_tool_set_ui_pad_start( 1 );
    sleep (6);


    while (r_control.start == 1) {
    //Taking Control Mutex
        vp_os_mutex_lock( &control_data_lock );
        r_control = control_data;
    vp_os_mutex_unlock( &control_data_lock );

    //command to make drone move
    ardrone_at_set_progress_cmd(0,0,0,r_control.gaz,r_control.yaw);

    prev_control = r_control;
    }

    ardrone_tool_set_ui_pad_start( 0 );
    PRINT( "Communication Thread Ending\n" );
    return C_OK;
}
```

The inter-thread communication is achieved by use of a mutex. Locking the mutex will deny all other threads until unlocked.

4.3.5 Simple Logic Statement Feedback

The simplest way to control the drone is by logic statements. Is the target position off to the right? Move the drone to the right.

Such a system will not allow the drone's movement to be proportional to the error, but rather slowly move in the needed direction until the position is within acceptable bounds.

An example of how this would work in the program is shown in Listing 4.10. Code is included for both displaying the coordinates of screen, and writing to file.

Listing 4.10: Logic Control Example (From within Processing Loop)

```
CvMoments *moments = (CvMoments*)vp_os_malloc(sizeof(CvMoments));
cvMoments(dst, moments, 1);
double moment10 = cvGetSpatialMoment(moments, 1, 0);
double moment01 = cvGetSpatialMoment(moments, 0, 1);
double area = cvGetCentralMoment(moments, 0, 0);
xposc = (moment10/area) - (width/2);
yposc = (moment01/area) - (height/2);
r=sqrt(area/3.1418);
    d = 100/r;
if (area > 150) {

    // Displaying coordinates on screen
sprintf( buf, "%.0f %.0f %.3f", xposc, yposc, d );
cvPutText(dst,buf,cvPoint(50,50), &font1,
    CV_RGB(255,255,255));

rposc = (r - 17); //Distance demand
if (rposc > 4) {rin = 0.3;} // Limits
else if (rposc < -4) {rin = -0.3;}
else {rin = 0;}

if (xposc > 50) {xin = 0.3;} //Yaw demand
else if (xposc < -50) {xin = -0.3;}
else { xin = 0;}

if (yposc > 50) {yin = -0.3;} //Height demand
else if (yposc < -50) {yin = 0.3;}
else { yin = 0;}
```

```
    // Locking Control Data Lock
vp_os_mutex_lock( &control_data_lock );
control_data.yaw = xin;
control_data.gaz = yin;
control_data.pitch = rin;
vp_os_mutex_unlock( &control_data_lock );

} else {
xposc = 0;
yposc = 0;
rposc = 0;
xin = 0;
yin = 0;
rin = 0;

    // Locking Control Data Lock
vp_os_mutex_lock( &control_data_lock );
control_data.yaw = xin;
control_data.gaz = yin;
control_data.pitch = rin;
vp_os_mutex_unlock( &control_data_lock );
    }

    oFile = fopen ("FRT_output.txt", "a");
fprintf( oFile, "%.0f %.0f %0.3f \n", xposc, yposc, d);
fclose(oFile);
```

4.3.6 Proportional/Derivative Control

PD control can be applied to the drone, using the positional feedback provided by the drone camera. PD control is a lot faster (with regards to physical reaction, not computational speed) and smoother than the logic-controlled system proposed above, the only disadvantage lies in the design process. Getting the gains right in the controller could take a lot of experimentation.

An example of how PD control would work in this program is shown in Listing 4.11. The only difference from 4.10 is how the drone command values are calculated. In this case, it is by a proportional and derivative formula with assigned gains as opposed to

logic control.

Listing 4.11: PD Control Example (From within Processing Loop)

```
CvMoments *moments = (CvMoments*)vp_os_malloc(
    sizeof(CvMoments));
cvMoments(dst, moments, 1);
double moment10 = cvGetSpatialMoment(moments, 1, 0);
double moment01 = cvGetSpatialMoment(moments, 0, 1);
double area = cvGetCentralMoment(moments, 0, 0);
xposc = (moment10/area) - (width/2);
yposc = (moment01/area) - (height/2);
r=sqrt(area/3.1418);
    rposc = r - 30;

if (area > 150) {
sprintf( buf, "%.0f %.0f %.3f", xposc, yposc, d );
cvPutText(dst,buf,cvPoint(50,50), &font1,
    CV_RGB(255,255,255));

//Distance demand
if((rposc < -5) || (rposc > 5)) {
ru = (rposc/80) + ((rposc_old - rposc)/30);
rin = rin_old + ((ru-rin_old)/40);
if (rin > 0.7) { rin = 0.7;}
else if (rin < -0.7) { rin = -0.7;}
} else { rin = 0; }

    // Yaw Demand
if((xposc < -50) || (xposc > 50)) {
xu = (xposc)/((width/2)) + ((xposc_old-xposc)/50);
xin = xin_old + ((xu-xin_old)/40);
if (xin > 0.7) { xin = 0.7;}
else if (xin < -0.7) { xin = -0.7;}
} else { xin = 0; }

    // Gaz
if((yposc < -50) || (yposc > 50)) {
yu = -(yposc)/((height/2)) - ((yposc_old-yposc)/50);
yin = yin_old + ((yu-yin_old)/40);
if (yin > 0.7) { yin = 0.7;}
else if (yin < -0.7) { yin = -0.7;}
 } else { yin = 0; }

sprintf( buff, "Yaw: %.3f, Gaz: %.3f, Pitch: %.3f",
    xin, yin, rin);
```

```
cvPutText(currframe,buff,cvPoint(50,100), &font1,
    CV_RGB(255,0,0));

} else {
xposc = xposc_old;
yposc = yposc_old;
rposc = rposc_old;
xin = 0;
yin = 0;
rin = 0;

}

    // Locking Control Data
vp_os_mutex_lock( &control_data_lock );
control_data.yaw = xin;
control_data.gaz = yin;
control_data.pitch = rin;
vp_os_mutex_unlock( &control_data_lock );
    }

    oFile = fopen ("FRT_output.txt", "a");
fprintf( oFile, "%.0f %.0f %0.3f \n", xposc, yposc, d);
fclose(oFile);
```

The gains level for the proportional and derivative components may need to be 'custom-fitted' to each task. The derivative gain may need to be scaled to match the proportional gain, due to the derivative feedback being calculated on the error difference between frames. This difference can be extremely small, depending on how fast the framerate on the camera is.

Additionally, there is an error threshold in every direction (for example, -50 < x < 50). This is to prevent the PD feedback from kicking in until there is a sizeable error. The reason for this is it has been found through preliminary testing that the drone tends to freeze up when constantly being given incredibly small output values.

4.3.7 Keyboard Command

The last thing that needs to be done before moving onto the homing task itself is making sure that commands can be sent from the user to the drone. The easiest way to do this is through the keyboard.

A simple command *cvWaitKey* will wait for any input from the keyboard, and then If() statements can be set up so that different drone commands can be triggered from different key inputs.

This principle in action is shown in Listing 4.12. The *cvWaitKey* command is called just after a frame is displayed[5].

Listing 4.12: Keyboard Input Example

```
C_RESULT display_stage_transform (display_stage_cfg_t *cfg,
vp_api_io_data_t *in, vp_api_io_data_t *out)

{
        char c=0;
    uint32_t width = 0, height = 0;
    getPicSizeFromBufferSize (in->size, &width, &height);

    IplImage *img = ipl_image_from_data((uint8_t*)in->buffers[0],
        1, 640, 360);

    cvNamedWindow("video", CV_WINDOW_AUTOSIZE);
    cvShowImage("video", img);
    c = cvWaitKey(20);
        cvReleaseImage(&img);

        // Copied code from red ball example for keyboard input
        if( c=='k' )
  {
    end_all_threads = 1;
    return( 1 );
  }
  else if( c == 'c' || c == 'C' )
  {
     if(videoChannel == ZAP_CHANNEL_HORI)
     {
```

[5]In reality, it could be placed anywhere in looping code.

```
            videoChannel = ZAP_CHANNEL_VERT;
            ardrone_tool_configuration_addevent_video_channel
            ( &videoChannel,NULL );

            vp_os_delay(500);

        }
        else
        {
            videoChannel = ZAP_CHANNEL_HORI;
            ardrone_tool_configuration_addevent_video_channel
            ( &videoChannel,NULL );

            vp_os_delay(500);
        }
    }

    return C_OK;
}
```

Chapter 5

Machine Vision Tests

This chapter will document the machine vision tests. These were performed early on in the year to verify the effectiveness of program to recognize objects in the incoming frames.

The tests will be found in the SDK directory as "DT1, DT2..." etc. This stands for "Detection Test".

5.1 First Test - Coloured Ball

5.1.1 Objectives

This test is to assess the drone's ability to find a pink ball (shown in Figure 4.1 on page 31). The colour pink was chosen as an easy colour to distinguish from a background, and being outside of the normal colour range seen both indoors and out, eliminating potential of background noise during tests.

The aim is to start the drone (without motors), move the ball around in front of the camera, and plot the ball's position as seen from the drone. The pre-defined path will

then be compared to the plotted path to verify effectiveness of the code.

5.1.2 Code Configuration

The program written for this test will use the following functions to process the image:

- *cvCvtColor(BGR2RGB)* - this will change the frame from the received BGR colourspace to RGB. This needed to display the frame properly on screen.
- *cvCvtColor(RGB2HSV)* - this will change the colourspace to HSV, in preperation for a binary search.
- *cvInRanges* - this function will perform the binary search, returning a single channel image with the result. The input parameters for this test were:
  - Hue: 125 to 135
  - Saturation: 120 - 255
  - Value: 60 - 255
- *cvMoments* - the moments of the binary image will be taken, so that the average pixel values in the x and y axis can be calculated. The total amount of detected pixels will be used to calculate the radius of the ball from the perspective of the camera. The distance can then be estimated from this.
- *cvCircle* - this will draw a circle over the RGB image showing the detected location of the ball.
- *fprintf* - the coordinates of the ball will be written to a txt file using the standard C file I/O library.
- *cvShowImage* - show frame.

The drone is programmed to begin recording data upon first detection of the ball, and at the 10 second mark, stop recording and save an image of the current view. If timed correctly by me, this will show the ball at its final position along the pre-set path.

The flow chart for this program is shown in Figure 5.1.

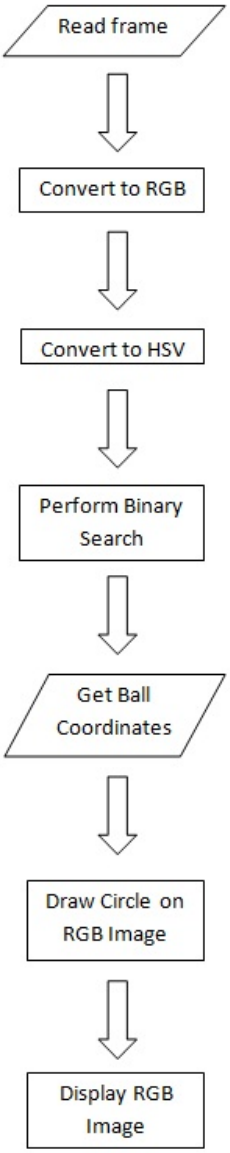


Figure 5.1: Detection Testing Flow Chart

5.1.3 Testing Location

The location for all indoor testing performed for this project is Z128, Z Block, USQ [1]. The room has a sizeable stretch along the far side of the room, suitable for all machine vision and feedback testing. However, the area may not be ideal for the full homing tests, so provisions or assumptions will be made to make the area useable if a better room isn't found in time.

Pictures of the room are shown in Figures 5.2 and 5.3.

Figure 5.2: Z128 Testing Room

5.1.4 Test Setup

The setup for this test involves two markers, once at the 1 meter mark away from the drone camera, and another at the three meter mark (shown in Figure 5.4). These will serve as waypoints in the test.

[1] This section will not be shown again in subsequent indoor tests.

Figure 5.3: Z128 Testing Floor

Figure 5.4: Detection Test Setup

5.1.5 Expected Results

I did two simple movements with the ball for this test. The first was a backwards movement from the 1 meter marker to the 3 meter marker, and the second movement is the exact reverse. Additionally, since the drone won't start recording data until it sees the ball, I will being the ball into view from the left, centreing it in front of the camera before I begin the two movements. Doing this also allows the drone to display that it can track the pixel position of the ball, although the distance calculation will be getting more attention in this test, simply due to the uncertainty of the formula[2].

If working correctly, the drone will track this path accurately and reflect this in the plots to follow.

5.1.6 Results

The recorded positional data is given in Figure 5.5. All results are given with the X and Y coordinates relative to the centre of the image. It is important to note that the Y axis is positive in the downward direction[3].

The data matches the expectations quite well. The entrance of the ball from the left of the camera can be clearly seen, and the Y position holds fairly steady while the observed radius goes from large to small and then back again. There is a small amount of noise in the image, reflected by sharp changes in the lines. Modifications could be made to further smooth the image, to eliminate such isolated pixels, but time didn't permit for me to perfect this.

The images taken at the end of the 10 second period are shown below the plot in Figures 5.6 and 5.7. The current estimated coordinates of the ball will always be written to

[2]This is because the formula is dependant on the amount of pixels seen by the camera as in the specified colour range, and this is variable with the lighting conditions of a particular environment.

[3]The standard origin for OpenCV is located at the top left of the image, with the X axis going positive to the right, and the Y axis going positive down the image.

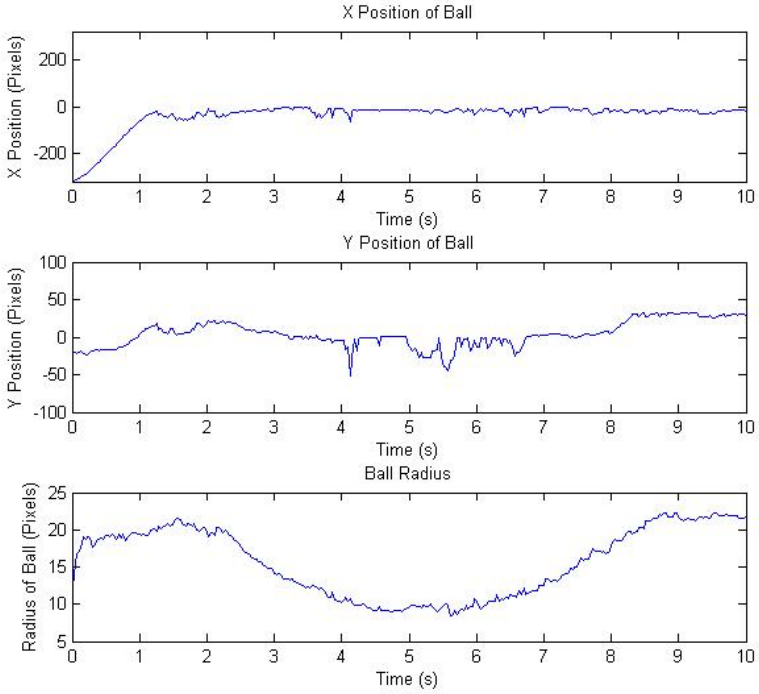


Figure 5.5: Ball Detection Test Positional Data

Figure 5.6: Ball Detection Test Finishing RGB Capture

the final RGB capture, and these values should match the end of the graphs.

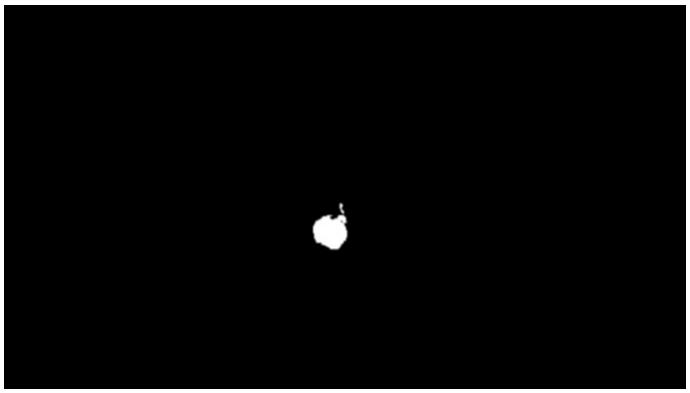


Figure 5.7: Ball Detection Test Finishing Binary Capture

5.2 Second Test - Hoop

5.2.1 Objectives

This test is to assess the drone's ability to find a hoop (shown in Figure 4.3 on page 34). This hoop will be functioning as an obstacle in later tests.

Once again, the drone will be running its camera only, plotting its interpretation of the object's position. The results will be compared to the expectations.

5.2.2 Code Configuration

This test will utilize the improved method of hoop detection (refer to Section 4.3.2).

The general structure of the program is the same as the coloured ball test, with the *cvHoughCircle* function replacing the *cvMoments* function to get the coordinates.

The arguements for the *cvInRanges* and *cvHoughCircle* functions are as follows:

- *cvInRanges*
  - Hue: 10 - 40

  - Saturation: 30 - 256
  - Value: 30 - 256

- *cvHoughCircle*
  - Minimum Distance Between Circles: 400
  - Canny Threshold: 300
  - Accumulator Threshold: 100
  - Minimum Radius: 5
  - Maximum Radius: 400

The drone is programmed to begin recording data upon first detection of the ball, and at the 10 second mark, stop recording and save an image of the current view. If timed correctly by me, this will show the ball at its final position along the pre-set path.

For the flow chart of this program, refer to Figure 5.1, but replace the "Get Ball Coordinates" with "Get Hoop Coordinates".

5.2.3 Test Setup

The same two markers used in the previous test will be used here (see Figure 5.4).

5.2.4 Expected Results

The planned path is identical to the previous test. The hoop will come in from the left, and then move from 1 to 3 meters, and back again. The plots will hopefully look similar to the results shown above, although the radius (r) value will be very different.

5.2.5 Results

The recorded data from the drone over the 10 second test is given in Figure 5.8.

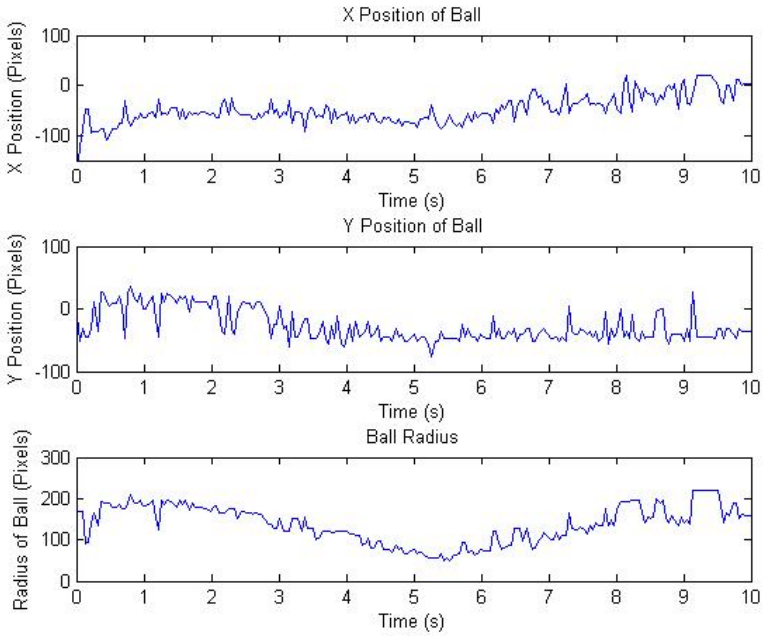


Figure 5.8: Hoop Detection Test Positional Data

Figure 5.9: Hoop Detection Test Finishing RGB Capture

The results of this test still follow the general pattern that was expected, but there is much more variation in all axes. The average X and Y readings maintain the expectations, but the frame-by-frame changes will likely cause the feedback controller to become erratic. This will be closely monitored in the later tests that involve the hoop.

Figure 5.10: Hoop Detection Test Finishing Binary Capture

5.3 Conclusion

Both of the detection tests have been successful in their intended tasks. The application in a larger program, however, still remains a concern. Since the detection methods are reliant on colour information, the performance of the code will vary with different lighting conditions, and different environments. Unfortunately, attempts (as part of this project) to work around this dependence have been unsuccessful (see Section 4.3.3).

With minimal amounts of noise already encountered in these tests, the nearby objects of similar colour will need to be removed from the room, to ensure that the remaining tests function properly.

Chapter 6

Tracking Feedback Tests

This chapter will document the drone tracking tests. These were performed to develop the drone ability to centre on a target.

The tests will be found in the SDK directory as "TT1, TT2..." etc. This stands for "Tracking Test".

6.1 First Test - Logic Feedback

6.1.1 Objectives

This test is to assess the drone's ability to track the pink ball used in the machine vision experiments[1].

[1] All of the tracking tests will be performed with the pink ball, as the hoop readings can be inconsistent, and we don't want the machine vision getting in the way of the feedback test.

6.1.2 Code Configuration

The program written for this test will use the functions needed to indentify the ball and its position (refer to the function list above), as well as the feedback pattern shown in Listing 4.10 on page 40.

The drone will be recording 30 seconds of data starting at the detection of the ball. An image will be taken at the end of the tracking time as well, just for display.

The flow chart for this program is shown in Figure 6.1. This flow chart is representative of all the tracking tests.

6.1.3 Test Setup

There are no markers or objects put in place around the drone for these tests. The ball is the only piece of equipment used.

6.1.4 Expected Results

With logic control feedback, the drone will not act on the error of the ball position until it reaches a certain threshold (set by me to be 50 pixels from the centre in each direction). The graph for the drone's movement will therefore be very jolted, with short bursts of movement. The positional error graph should be smooth, as in the detection tests, but will change rapidly in response to any movement by the drone.

6.1.5 Results

The flight data for the test (separated by axis), is shown in Figures 6.2 through 6.5.

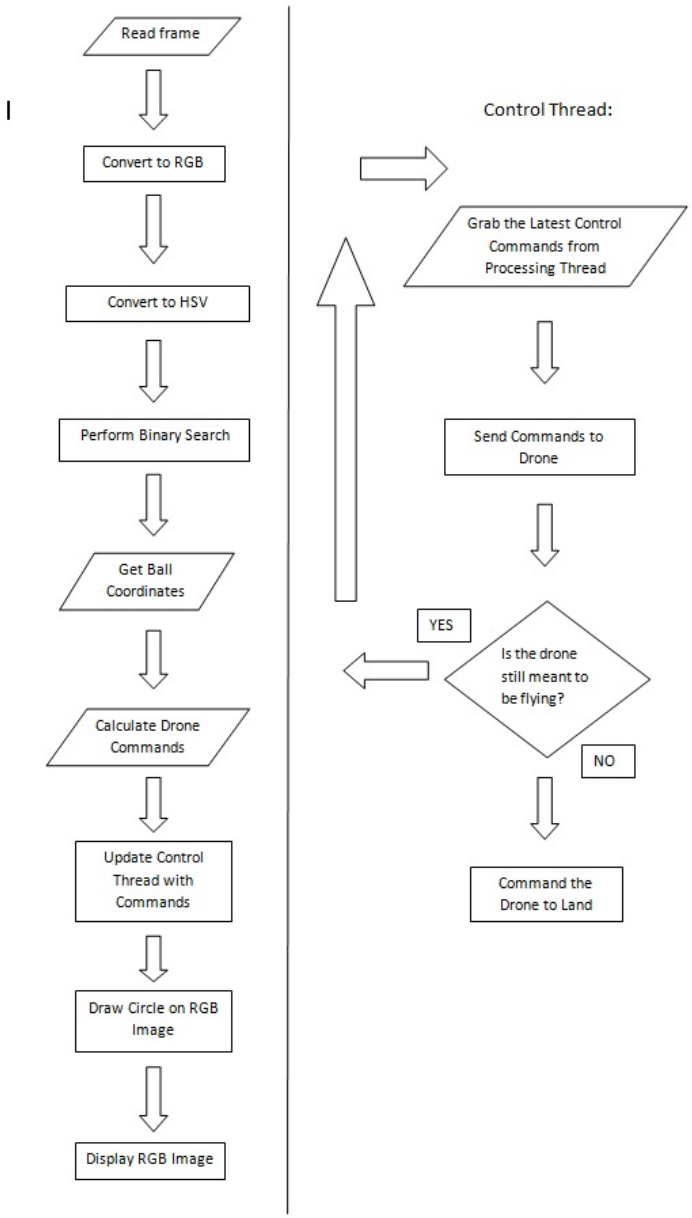


Figure 6.1: Tracking Test Flow Chart

The response from the drone matches the intended directions to counter the errors in each axis. In the X axis, an error to the right requires the drone to turn to the right to face it. In the Y axis, an error in the positive direction (the ball being too low) must see the drone lower itself to match. This is why the directional response in the Y axis is inverted, compared to the others. The R axis (labelled so for Radius), is non-inverted, with a lower radius needing a forward movement (a negative command value).

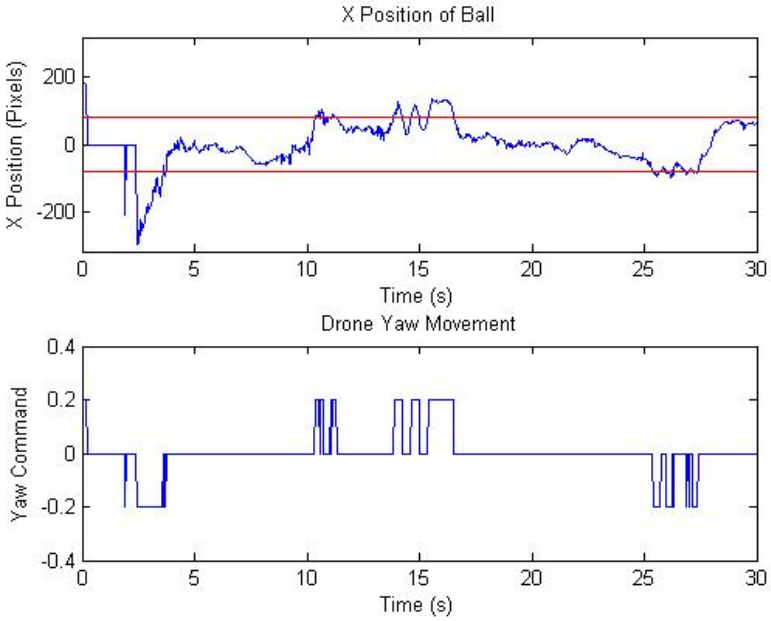


Figure 6.2: First Tracking Test X Axis Flight Data

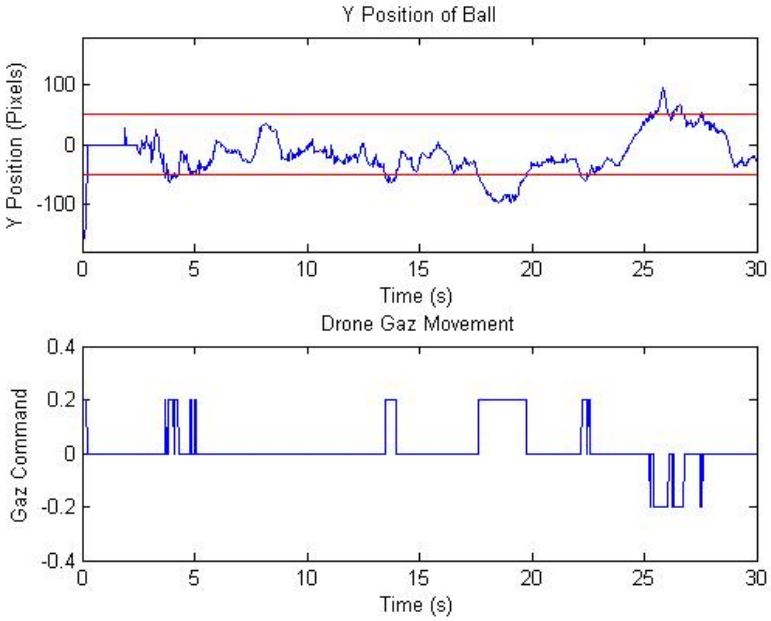


Figure 6.3: First Tracking Test Y Axis Flight Data

The feedback values seem to be getting across the threads fine, and the drone is responding to errors in each axis beyond the acceptable range. The drone jolts around a

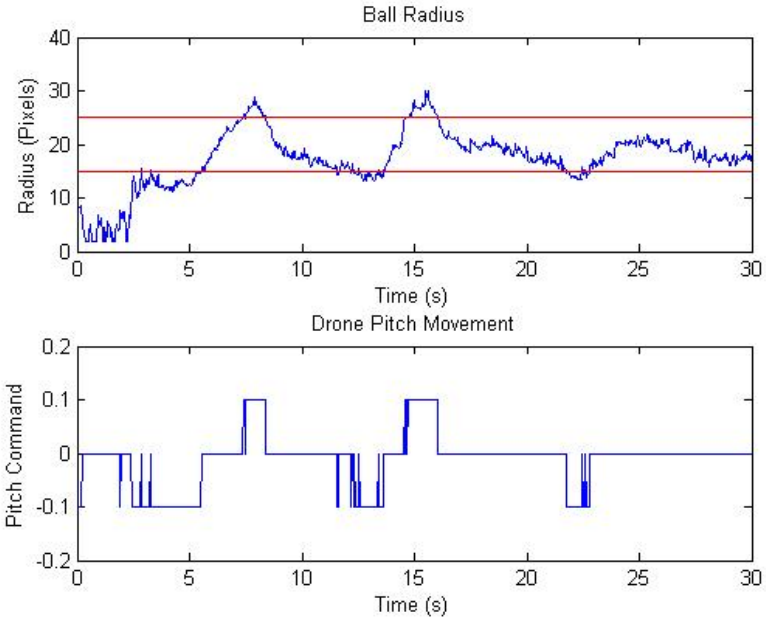


Figure 6.4: First Tracking Test Radial Axis Flight Data

Figure 6.5: First Tracking Test Finishing RGB Capture

bit, though. Other methods of feedback will be able to address this.

6.2 Second Test - PD Control

The last two tests of this chapter were not performed at Z128. In doing some last minute changes to the code, I found the need to re-run the test, but had no access at the time to the lab. Instead, this test has been performed just in a living room. This won't have any negative impacts on the test, given that it is close range.

6.2.1 Modifications

This test run will implement the PD control method developed and shown in Listing 4.11 on page 42.

The objectives, code setup, and testing setup are all near identical to the first tracking test, with the difference being the feedback method used to calculate needed drone movement.

The specific gains used in this test(labelled P for the proportional gain and D for the derivative gain) for each axis are:

- XP:1/320
- XD:1/50
- YP:1/180
- YD:1/50
- RP:1/80
- RD:1/30

Since the error is expressed in pixels, and the maximum directional drive is 1, these fractions are necessary to bring the feedback values into an acceptable range.

6.2.2 Expected Results

In contrast to the first version of the test, this run should present a much smoother feedback graph. All other results should be similar to above.

6.2.3 Results

The flight data for this test is shown in Figures 6.6 through 6.9.

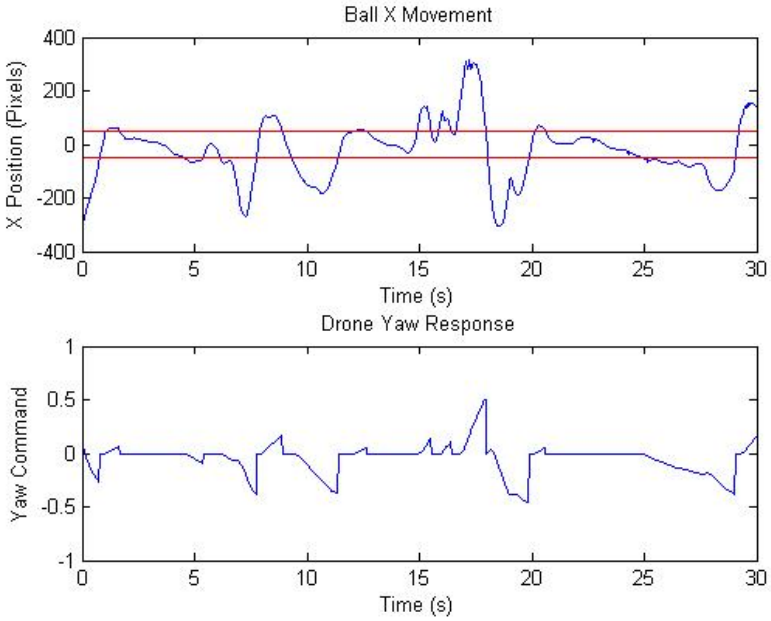


Figure 6.6: Second Tracking Test X Axis Flight Data

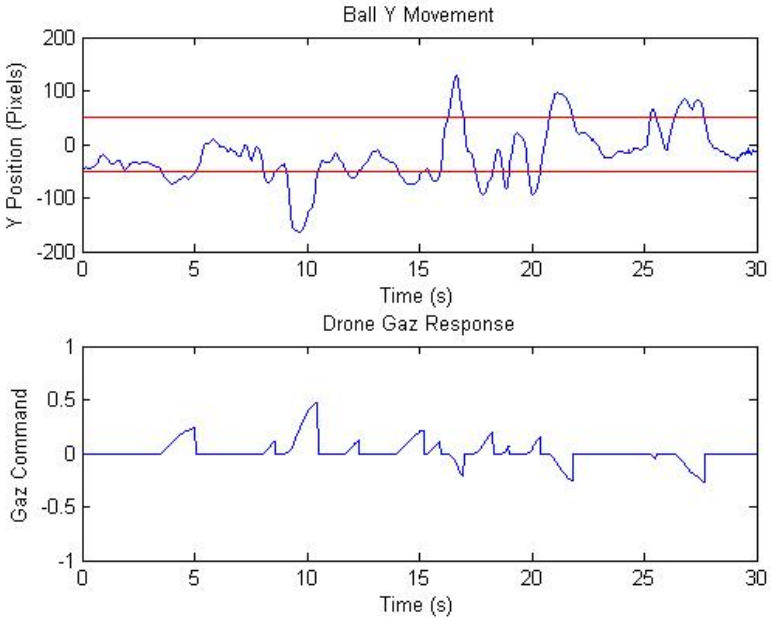


Figure 6.7: Second Tracking Test Y Axis Flight Data

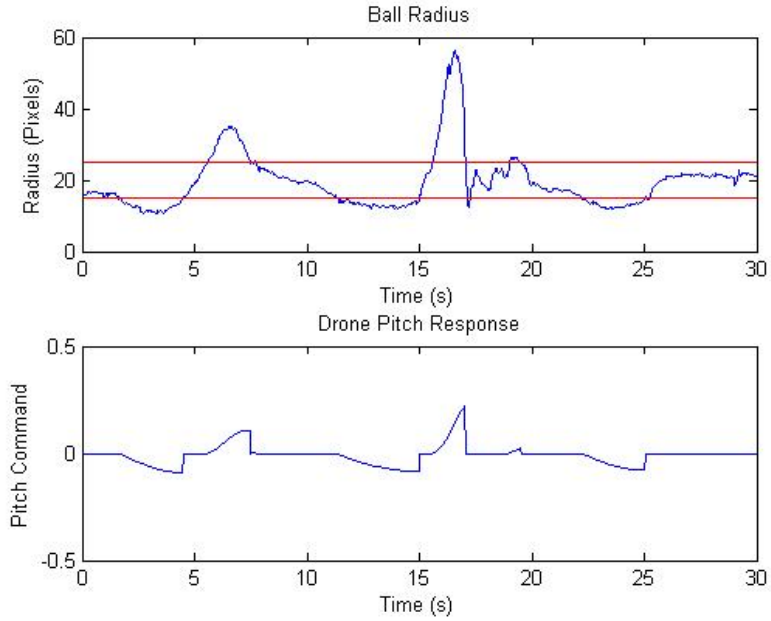


Figure 6.8: Second Tracking Test Radial Axis Flight Data

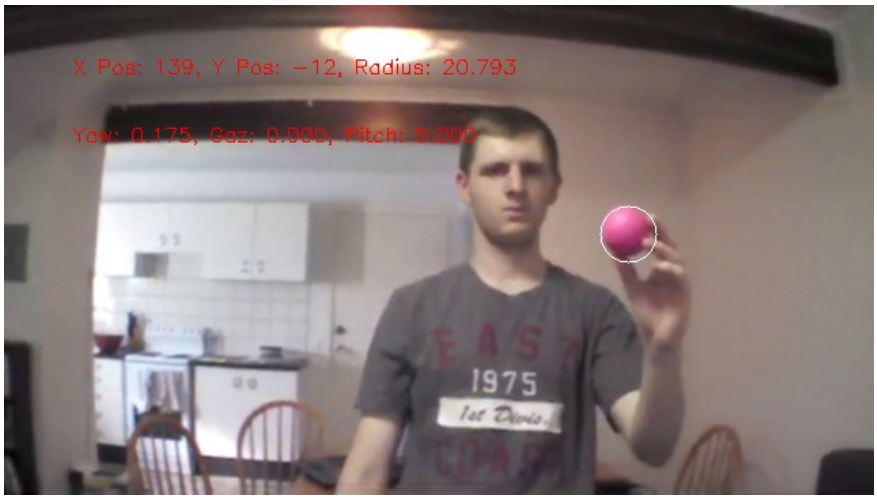


Figure 6.9: Second Tracking Test Finishing RGB Capture

As expected, the PD feedback kick in when the drone leaves the positional limits. The lines themselves, though, do not appear to be representative of typical PD control. This could be for any of the following three reasons:

1. The initial error that is required for the PD control to kick in halts the feedback before the curve can be seen.
2. The change from one feedback value to the next has been defined as gradual.
3. The derivative gain values isn't properly calibrated yet.

If time permits, further testing will be done to optimize the gain values. This will improve the performance of the drone in these experiments greatly.

6.3 Third Test - Bottom Camera PD Control

6.3.1 Modifications

This test run will implement the PD control method once again, but using the bottom camera of the drone, to maintain a central position over the ball.

The only other difference to the code used in the previous PD control test is the variables to which the feedback calculations are writing. Previously, the *yaw* and *gaz* variables have been used to maintain horizontal and vertical positioning, respectively. In this test, the *roll* command will maintain horizontal positioning, and the *pitch* command will move the drone forward and backward, in pace with the movement of the ball.

In addition, lower gains will be set for this test run, as it has been found (through experimentation too embarrassing to document here) that the drone will overshoot the target badly, if using the gain values from the previous test. The pixel error limits may also be raised, to prevent the drone from reacting too quickly.

Gains:

- XP:1/2560
- XD:1/200
- YP:1/1440
- YD:1/200
- RP:1/80
- RD:1/200

6.3.2 Expected Results

The flight data for this test run should not be too different to those of the previous test, since the same method for calculating flight values is being used.

6.3.3 Results

The results are shown in Figures 6.10 through 6.12. No end of test capture was taken for this test, as the drone did not have sight of the ball at the time.

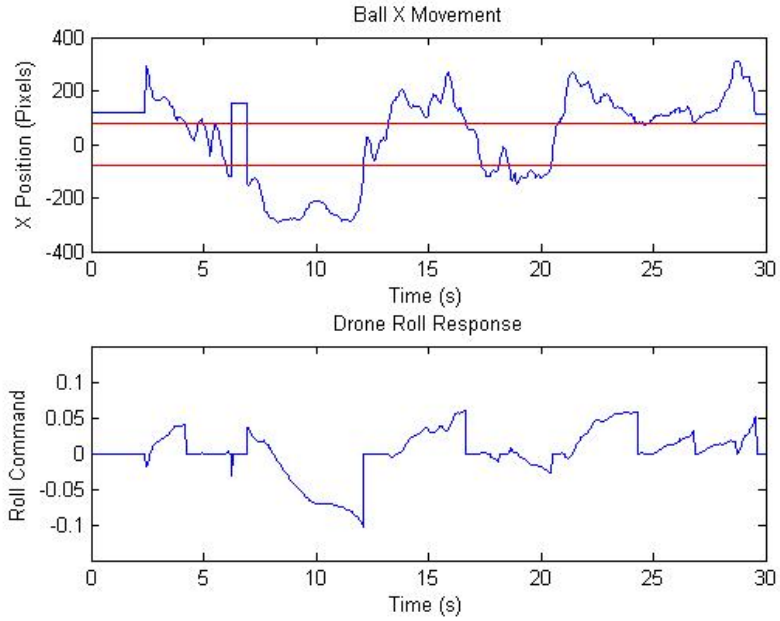


Figure 6.10: Third Tracking Test X Axis Flight Data

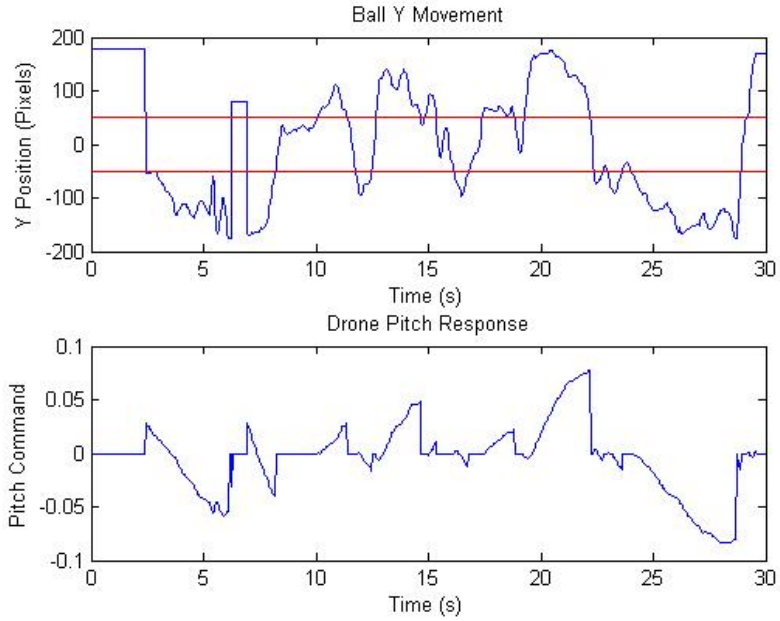


Figure 6.11: Third Tracking Test Y Axis Flight Data

The lower gain levels appear to be suitable for this camera. The only observation to be made of this data set is that the distance/radius graph shows that the drone went

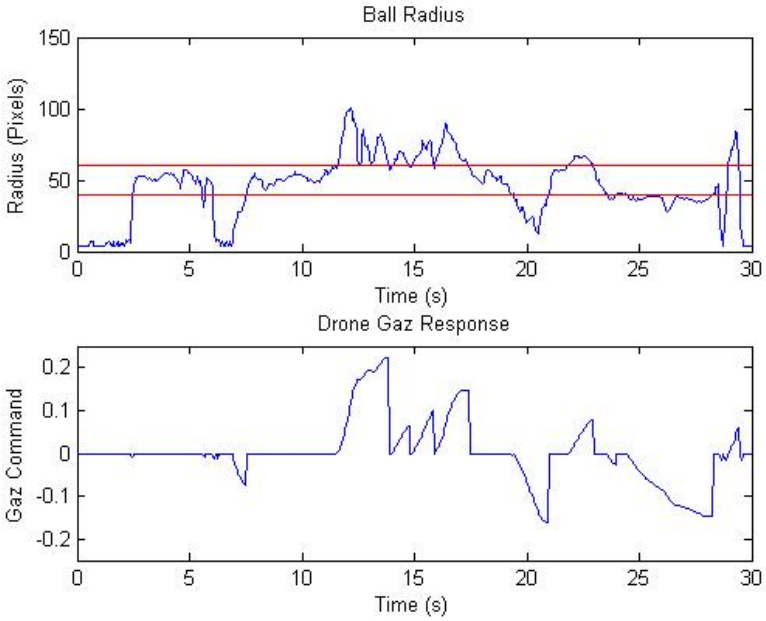


Figure 6.12: Third Tracking Test Radial Axis Flight Data

up far more than it went down. This was due to the drone starting near the ground, and me forcing it up to standing height.

6.4 Conclusion

Each method of feedback proved effective in following a moving ball. The PD control seemed to achieve this more smoothly, and will be used in the homing tests (to follow) wherever possible. The logic-controlled system is more prone to overreacting if the feedback gains are set too high, or under-reacting and losing the target, if set too low. A PD controller will be more capable of handling any all error sizes.

It was also discovered that the top-down PD feedback needed to be more sensitive than the front-facing configuration. The lower gains used in third test of this set will need to be brought over to any subsequent tests.

Chapter 7

Drone Homing Tests

In this chapter, the final implementations (the tasks matching the project specification) will be covered. Each will take different features of the developed functions, and combine them to achieve a specific goal. The tests are now progressive work, each program is being built using the previous program as a base. This is starting from the first homing test here, being built from the final tracking test.

The tests will be found in the SDK directory as "HT1, HT2..." etc. This stands for "Homing Test".

7.1 Input/Output Comparison

Before the homing tests are examined, the effects of the drone's flight behaviour will be noted.

Throughout the tests, the A.R. Drone hasn't acted exactly as the inputs would dictate. Both idle hovering and movement are inconsistent[1] with the drone, and this has had a great effect on aspects of the tests, specifically sections where the drone is "flying

[1]Unfortunately, the errors have been too inconsistent to calibrate around.

blind", without any vision guidance.

To demonstrate this, the Figure 7.1 shows the drone's view of the pink ball. The code from the first detection test (DT1) is being used, but the drone has been put in the air, rather than sitting on the ground or on an elevated platform.

There has been no commands given to the drone, so it is attempting to hover on the spot. It is therefore expected that the ball position (if kept stationary) will remain fairly constant. The result, however, shows the drone moving away from the stationary ball.

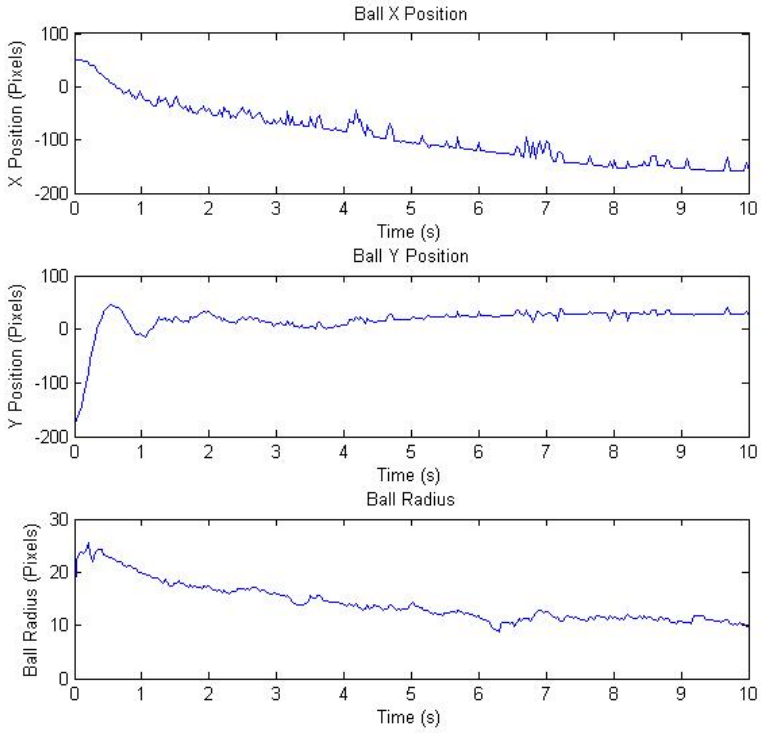


Figure 7.1: Drone Hover Test

Attempts were made to re-calibrate the drone, using the official drone applications, but with no effect. The impacts on the following tests will be mentioned where significant, but will be tried to be avoided.

7.2 First Test - Bottom Camera Homing Test

7.2.1 Objectives

This test is to try and implement the most basic version of a homing function. The drone will be manually searching an area for the ball, and will centre itself over it once found. The bottom camera of the drone will be used for the entire test. It will therefore have no vision of its path, and will be relying on the precision of the person giving the input.

7.2.2 Code Configuration

To perform the full homing task, several stages will be involved. The drone first has to search, and then track. To handle this, case code will be introduced into the program, allowing the code to step through the various stages of the process. Each stage will have conditions that need to be met before the next stage will be triggered. The stages for this test and their conditions to proceed are:

1. Hovering - the drone will hover on the spot, until given keyboard input from the tester, signalling it to begin. The keyboard input code will directly alter the case variable, starting the process.

2. Searching - the drone will perform repetitive movements, in order to move over an area, in search of the ball. This stage can have mini-stages within it, running set movements based off of timers. This case code allows for any kind of open-loop search path to be programmed into it. The system used in this test will put out a square wave on the pitch plot, indicating jagged advancement forward. Once the ball has been located, the next stage will trigger.

3. Homing - the drone will centre itself over the target, using the PD control method tested in TT3. If the drone loses vision of the ball for any reason, it will start

to gain height in an effort to re-establish visual contact. Once the drone has maintained an acceptable position over the ball for a few seconds, the program will proceed to the next stage.

4. Landing - the drone will move in a specified direction for a second or two, to get away from the ball, and then land.

The gains and pixel error limits are the same as from TT3.

The drone will begin recording flight data as soon as the first stage begins (the hovering stage counts as stage 0). It will continue recording until the landing stage. In addition, the normal screen overlay will be re-implemented, and an image capture of the final hovering position of the drone will be taken.

The flow chart for this program is shown in Figure 7.2. This flow chart is representative of all the homing tests.

7.2.3 Test Setup

The path for this test will just be straight forward, with the ball directly ahead of the drone (shown in Figure 7.3 and 7.4). The dustpan is being used as a cradle for the ball, as the drone's generated air flow has moved the ball around in preliminary testing.

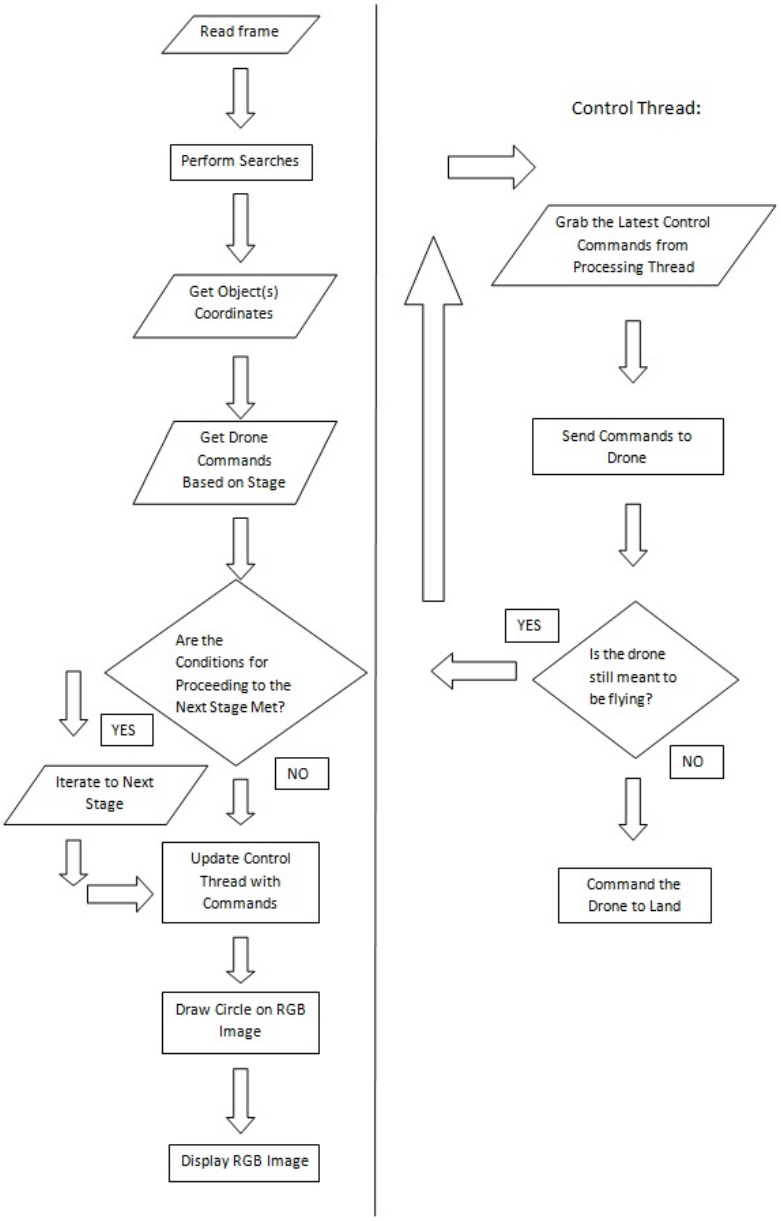


Figure 7.2: Homing Test Flow Chart

Figure 7.3: First Homing Test Setup

Figure 7.4: First Homing Test Ball Placement

The following assumptions were put in place to help achieve the needed result in the test without sacrificing the effectiveness of the code operating under normal conditions:

- A straightforward path is sufficient for use in this test, as it still uses and proves the effectiveness of the case code that has been written to cater for any search path.
- The initial hovering stage of the drone can be kept to be as short as possible in order to prevent drifting, without effecting the result.
- The drone can be twisted as necessary to correct its "forward path" due to drifting[2].
- There will be no altitude challenges in the test. The ball will be resting on the same level as the drone at rest.
- There will be no lighting or environmental challenges to make the colour-searching difficult.
- The ball can be placed quite close to the starting location. This is to prevent the image processing section of code from ruining the testing of the tracking function and case code.

The only assumptions that I believe reflects negatively on the written code are the last two. Whilst it has been part of this project to develop effective image processing to detect the ball, the testing of the other components of the code is a priority, in this instance.

[2]This assumption has been made because I believe it would be better to cater for the uniqueness of the drone without altering code, tailoring it for a specific drone instead of being able to be used on any drone.

7.2.4 Expected Results

The expectations of this test are for the pre-programmed stages of the code to fully execute. The conditions have been set so that failure to either find the ball or stably centre over it will result in incompletion, and therefore, a failed test.

7.2.5 Results

The output data from the test run is shown in Figures 7.5 through 7.8.

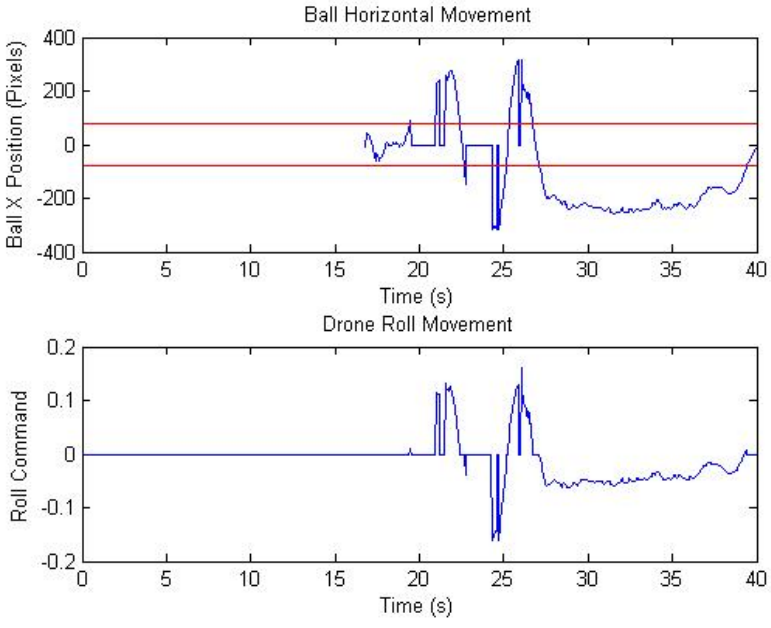


Figure 7.5: First Homing Test X Output

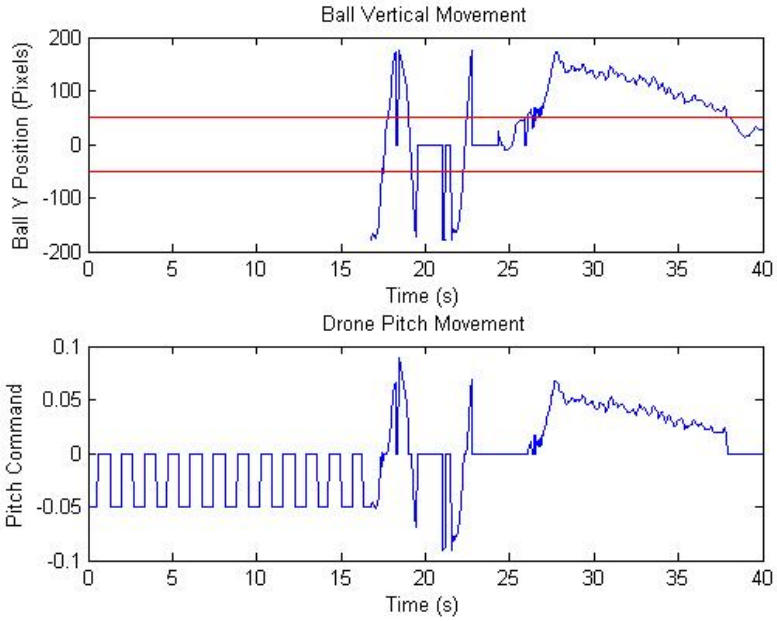


Figure 7.6: First Homing Test Y Output

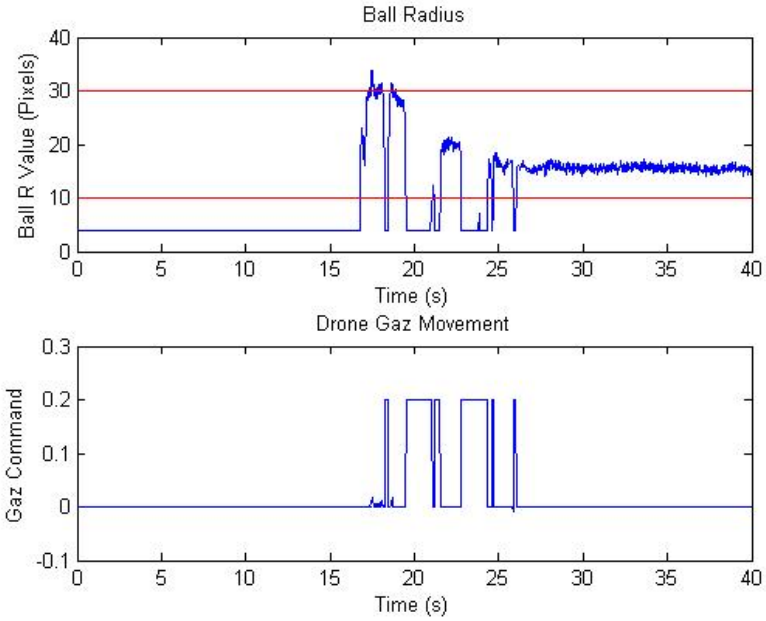


Figure 7.7: First Homing Test R Output

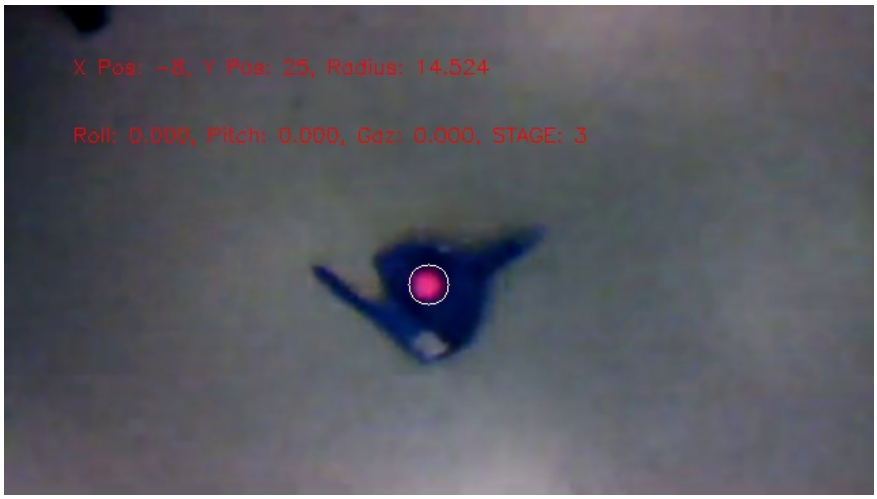


Figure 7.8: First Homing Test Finishing RGB Capture

It is pleasing that despite some issues moving a long a straight path, the drone still managed to find the ball, and eventually hover over it. It did take some time to fix the small error that was present in the position. This may be due to the PD controller not being fully tuned yet, or it could be just that the drone's response from such a height was simply slow.

All of this code will be used in the next iteration of the experiment. The PD control and structure of the case code will remain identical in both of the remaining tests.

7.3 Second Test - Full Homing Task

7.3.1 Objectives

The next test will build on the searching methods used in the previous test. Instead of blindly searching an area with the belly camera, the drone will now search its immediate area with the front camera, position itself in front of the ball, and then switch cameras and make a short movement to get above it in order to centre over it, as before. This will give the drone the ability to find the ball before it even starts moving (given that no obstacles are in the way to block line of sight). If compared to the previous method

for a search area that extends 360 degrees around the starting position, this method will prove much faster, as it is finding the direction to set off in, as opposed to having to cover it all.

7.3.2 Code Configuration

This new function specification will add more steps to the case code. The case code was developed over a lot of preliminary testing, and the version to be used in the final test is as follows:

1. Hovering - as before. Keyboard signals are still being used to break out of this hover mode, and begin the program.
2. Move Down - the drone will move downwards, bringing itself closer to the ground in order to more easily spot the ball with its front camera. Once the timed movement has been completed, the code automatically proceeds to the next stage.
3. Searching - the drone will rotate on the spot, looking for the ball. This stage will end when the ball has been located, and it is positioned fairly centrally in front of the cameraThe limit was set to 50 pixels in either direction, considering the X axis only.
4. Forward Movement - the drone will proceed towards the ball. If the horizontal error gets too big, logic based feedback will bring the drone back into line. The case code will progress once the drone has lost vision of the ball out the bottom side of the image view.
5. Timed Forward Movement - the drone will move forward for another short time period, and the camera will switch to the belly camera.
6. Timed Upward Movement - another timed movement to get the drone higher in the air to more easily track the ball.
7. Tracking - the same function from both HT1 and TT3.

8. Landing - same again.

Both the PD feedback gains and the flight data recording conditions have been kept the same from the last test. Screen captures will be taken, though, at the end of any stage that has vision of the ball, to provide more detailed information about the flight.

7.3.3 Test Setup

The starting position of the drone and the sitting position of the ball are similar to the above test (refer to Figure 7.3). The drone has been twisted in the other direction for this test, though.

Assumptions:

- The initial hovering stage of the drone can be kept to be as short as possible in order to prevent drifting, without effecting the result.
- The ball can be placed quite close to the starting location. This is to prevent the image processing section of code from ruining the testing of the tracking function and case code.
- There will be no altitude challenges in the test. The ball will be resting on the same level as the drone at rest.
- There will be no lighting or environmental challenges to make the colour-searching difficult.

Less assumptions are needed this time around. This is by deliberate design, as the point of the developments in this version of the code was to make the program less dependent on this assumptions.

7.3.4 Expected Results

Once again, the hope is for the code to execute all stages successfully.

For this test, a hand-guided flight was performed to take sample images to show the planned execution of the program in flight. The drone will still log all image data and outputs to the motors, the motors are off, allowing me to control the drone's movement and ensure that the test will theoretically be successful if the drone's movement as dictated by the code matches my hand-held movements.

The images were taken from the end of stage 2 (Figure 7.9), through to just before landing (Figure 7.12).

Figure 7.9: Pre-Second Homing Test Stage 2: Ball Located

Figure 7.10: Pre-Second Homing Test Stage 3: Forward Tracking

Figure 7.11: Pre-Second Homing Test Stage 6: Ball Re-Located from Above

Figure 7.12: Pre-Second Homing Test Stage 7: Homing Complete

7.3.5 Results

The plots and images to follow show the results from the test flight.

For the X position plot, stage 3 is triggered when the ball comes within the error threshold. This means that once the position enter the window (shown with red lines), the feedback commences.

For the y position plot, the value should just increase once the program enter stage 3 (as is seen here). The drone eventually moves close enough to the ball that it loses sight, triggering the next stage.

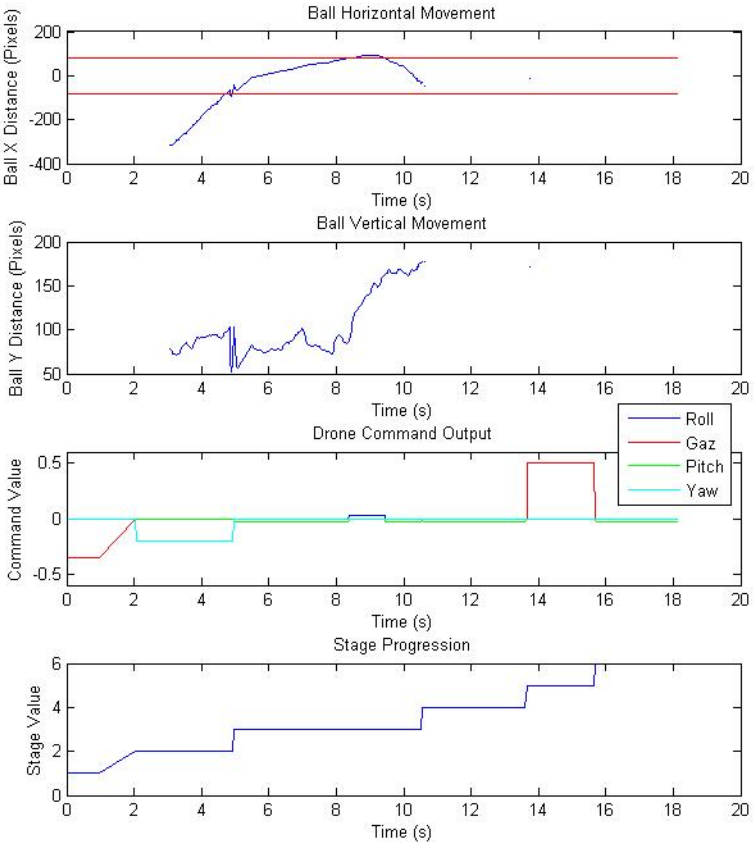


Figure 7.13: Second Homing Test: Flight Data

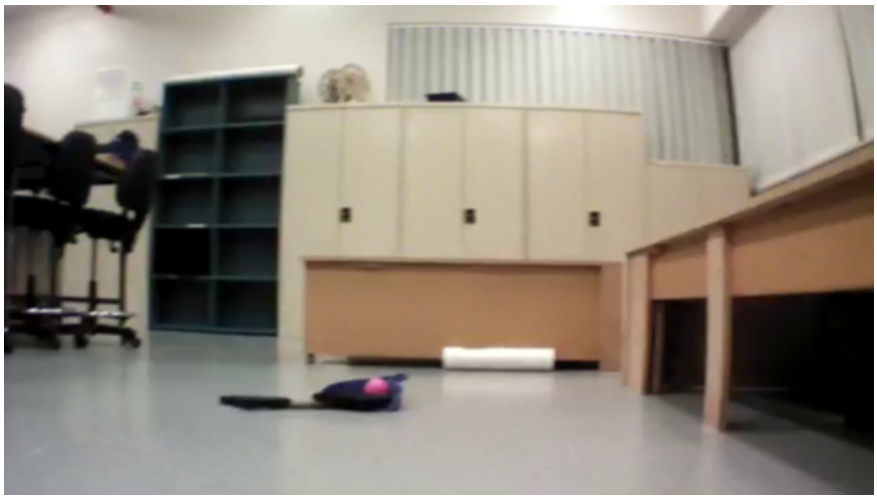


Figure 7.14: Second Homing Test Stage 2: Ball Located

Figure 7.15: Second Homing Test Stage 3: Mid-Stage 3

The plots show that the drone was unable to proceed to Stage 7, meaning that it was unable to locate the ball with its bottom camera. The drone therefore got stuck in stage 6 and continued flying forward, until manually stopped. The process up until stage 6 appears to have gone smoothly, with the drone performing all pre-programmed movements, and moving towards the ball with horizontal feedback in stage 3.

It has been confirmed with manual testing (running this same test again, but with manual stepping through by keyboard controls) that the camera switch function will

only change the camera when the drone is not flying. Firing this command off in flight, either by keyboard event or through the case code, will add the event to the queue, which will trigger as soon as the drone is told to land. The reason that the event is unable to resolve in mid-flight was not discovered after a few hours of testing the function, and so it has remained unsolved at the time of submission for this dissertation. Without the capability of switching cameras mid-flight, this test will not be able to be completed.

7.3.6 Discussion

Despite a failed test, there stages that did get to execute appears sucessful and accurate. The drone was able to stay on track when heading to the ball, and it the tracking function that would execute if the drone was able to find the ball again once it switched cameras has already been tested and proven to be functional. The only uncertainties that are left untested are the two stages that will try to get the drone into position above the ball. Adjusting these would be a simple matter, although calibration may be needed, as the drone would be flying without positional feedback, which can lead to drifting.

7.4 Third Test - Obstacle Avoidance

7.4.1 Objectives

This final test will look at combining the two image processing methods, and utilize positional awareness of both the ball and the hoop to perform a homing run with the hoop as an obstacle, that must be passed through en route to the ball.

All images shown below are of hand-guided flights. Time did not permit me to polish this code to the point of being test-ready (even if I did, I'd still have the issue of camera-changing mid-flight). However, just through the previous tests, a lot of the underlying

code for this task has already been written, and so it's just the case code that would need to be written and tested. In this report section, I will outline my proposed case code, and using previous test results to analyse the code to see how much of it would already be dependably useable, and how much is unsafe.

7.4.2 Code Configuration

For the first time, both image processing methods are being used in the same program. And whilst the circle detection algorithm can be switched off for most of the flight, it does hugely increase the complexity of the program. There nearly doubles the normal number of variables, making data harder to handle (at least, if not properly labelled).

There will also be more stages needed in the program, to cope with the increased complexity of the task. A proposed stage list is given below.

1. Hovering - as in previous programs
2. Move Vertical - the drone will move vertically, bringing itself to an approximate elevation to start searching,
3. Searching - the drone will rotate on the spot, looking for the ball. This stage will end when the ball has been located, and it is positioned fairly centrally in front of the camera[3]
4. Forward Movement - the drone will proceed towards the ball. If the horizontal error gets too big, logic based feedback will bring the drone back into line. The case code will jump to the Forward/Camera Switch case once the drone gets close the ball. As a priority (over the jump just afore mentioned), the case code will progress to the next case if the hoop is detected.
5. Hoop Positioning - if an obstacle hoop has been found, the drone will centre itself facing the hoop preparing to go through it. The case code will progress once its

[3]The error limit for this remains at 50 pixels in size.

satisfied that the drone is in a position to progress through the hoop. This can be determined using the hoops coordinates (x,y,radius) to give positional feedback to an ideal position, much the same as the drone has been tracking the pink ball.

6. Timed Forward Movement - the drone will go forward, through the hoop. This can be a timed movement, if the positional feedback from earlier placed the drone at a set distance.

7. Turn (Optional) - based on the balls position when the drone detected the hoop, the drone will make a turn to face the general direction that the ball is expected to be in. This case will rely on calculations made earlier in the flight.

8. Timed Forward Movement - the drone will move forward for another short time period, and the camera will switch to the belly camera.

9. Timed Upward Movement - another timed movement to get the drone higher in the air to more easily track the ball.

10. Tracking - the same function from both HT1 and TT3.

11. Landing - same again.

Again, the PD feedback for the top-down ball tracking has been kept the same, along with the data-logging conditions (0 < stage < landing). The feedback for the hoop, however, will likely have different gains if based on PD control, especially when dealing with the large radius of the hoop.

7.4.3 Test Setup

Since a proper test was not conducted, there are no starting positions and initial conditions to be documented in images. However ,the assumption list for the theoretical test can be still be specified.

Assumptions:

- The initial hovering stage of the drone can be kept to be as short as possible in order to prevent drifting, without effecting the result.
- The ball and hoop can be placed quite close to the starting location. This is to prevent the image processing section of code from ruining the testing of the tracking function and case code.
- There will be no lighting or environmental challenges to make the colour-searching difficult. Less assumption are needed this time around. This is by deliberate design, as the point of the developments in this version of the code was to make the program less dependent on this assumptions.

7.4.4 Expected Results

A hand-guided test run was performed, to step the drone through the current version of the case code. The logged images are shown in Figures 7.16 through 7.19

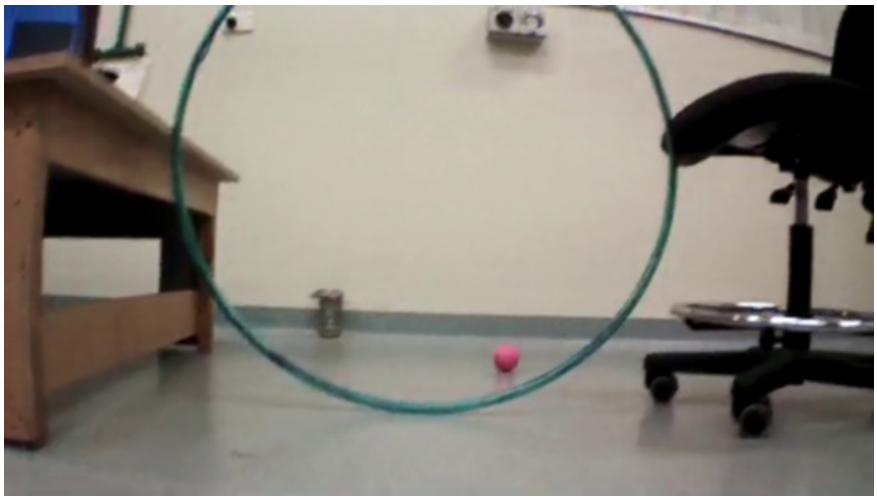


Figure 7.16: Third Homing Test Stage 2: Ball and Hoop Detected

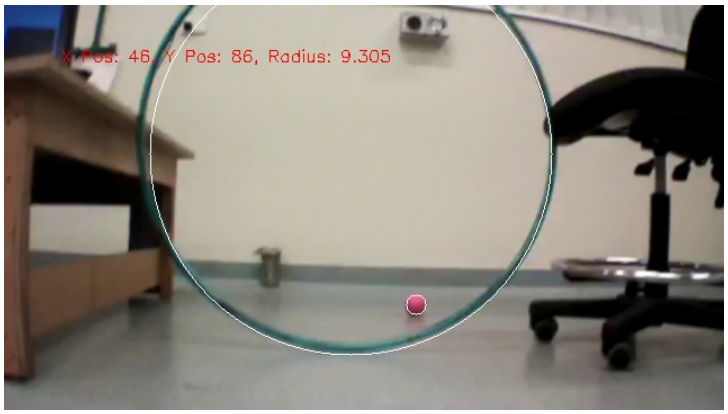


Figure 7.17: Third Homing Test Stage 3: Hoop Lined Up

Figure 7.18: Third Homing Test Stage 8: Ball Re-Located from Above

Figure 7.19: Third Homing Test Stage 9: Homing Complete

7.4.5 Theoretical Results

The purpose of showing this theoretical test is to bring to discussion the parts that are currently preventing this program from being functional. For the previous test (SHT), this was the drone switching and the blind flight. In this test, those two issues still both exist. The amount of time that the drone is "flying blind" is actually significantly increased, and would have caused problems for the testing, to the point where it could probably not be done without being addressed (refer to Section 8.2 on page 92).

The other untested factor that lies in this program is the hoop positioning stage. For the purposes of the theoretical flight, the case code served its purpose. But in proper testing, the current tracking code may not function well enough, or the conditions for defining the drone as "in position" to go through the hoop might be too strict, or not strict enough.

Whilst these elements remain in the way of this tasks completion at the time of submission of this dissertation, most of the work behind the program has been done, and it's just implementations of these techniques that need to be tweaked.

7.5 Conclusion

The full homing tasks, unfortunately, were not wholly completed. The test that did fully work, along with the partial flights and simulations, have still shown positive results. The drone is capable of using its bottom camera to centre over a target, and using its front camera to search and navigate. Whilst the obstacle detection feature wasn't fully implemented, a suggested base program has been proposed, and proof has been given that the drone is certainly capable of navigating through or around the hoop, given the proper instruction in programming.

The drifting of the unguided drone and the camera switching issue got in the way of the these test's completion. Further analysis of these problems is made in Section 8.2.

Chapter 8

Results and Discussion

The final summaries and findings of the project will presented in this chapter, along with all suggestions for future work on the A.R. Drone.

8.1 Project Outcome

The outcome of the project has been slightly underwhelming. A lot of time was lost in the testing phase, due to tweaking and debugging. The low flight/recharge ratio of the drone battery meant that a lot of time was spent waiting for the battery to recharge after changing as little as one line in a program's code.

The primary goal to the project specification – to create a homing function for the A.R. Drone – was completed. This created functionality was demonstrated in test HT1 (page 71), and expanded in HT2, which were built from information gathered during the preceding tests.

The optional goals, however, were not fully addressed. Test HT3 unsuccessfully attempted to meet one of the secondary objectives (obstacle avoidance), but was unable to be completed in the timeframe. Other secondary objectives such as elevation chal-

lenges and homing 'beacon' variants (such as those not recognizable by colour) were touched on, but not deliberately explored and designed for.

On the positive side, all testing that was done was able to be well documented and detailed, thanks to the ability to record flight data to file. The screen captures (and video, if it was added to the program) made for great visual display of the information, but were not as helpful as the data log in dissecting the performance of a test run. The plots that were made gave ample performance information, allowing the first two chapters of experiments (those focusing on basic drone functions) to have a form of numerical comparison with expectations. These plots didn't offer as much for the homing tests, but still provided proof of accomplishment of processes within the program, such as completing timed commands and responding to machine vision input.

The only disappointing aspect in the documentation for some of the experiments was the assumptions list. For the final homing tests, certain assumptions were made that made the conditions rather simple, with the intent of giving the later stages of the program a better chance of executing, and therefore, being documented, assessed, and optimized. It must be conceded that one of the goals of this project was to design machine vision that would be more robust, and therefore, more resistant to tougher conditions. However, the program had to make use with what was developed in the time frame.

8.2 Recommendations for Future Work

This section is for thoughts on the next steps for various topics related to this study.

8.2.1 Drone Optimization for Testing

There are a couple of things that I would recommend to anyone using the A.R. Drone in future research. On the hardware side, the A.R. Drone battery proved to be a

big issue throughout the testing phase (as mentioned above). Any improvement to this system will make experimentation a much faster process in the future. The two possible solutions for this are:

1. Purchase a more or higher capacity branded batteries from Parrot. This is the simpler solution, and will just mean one can have one recharging while another is being used, with the ability to swap them in and out.
2. Change out the battery with another brand of battery that can fit into the drone. In order to do this, the battery connector will have to be stripped and changed to suit the new battery. The benefit of this is access to other brands of batteries and rechargers, as the batteries from Parrot are widely held to be poor quality.

On the software side, two functions that would have made this project a lot better (if they worked) were the drone's camera switching command, and the OpenCV code to record all incoming frames to a video file. It is highly recommended that these functions be implemented, for the improvement of both the capabilities of the drone, and the capability of sharing the results of the research.

8.2.2 Machine Vision

The development of the machine vision for this project fell short of the initial expectations. Histogram backprojection was attempted, to remove the drone's colour dependence, but to no success. This suggested code could definitely be a starting point for someone to pick up the code and finish it off. The use of this method of detection opens a lot of possibilities when looking at object tracking, and navigation.

8.2.3 PD Control

Proportional/Derivative control was trialled and analysed in this project, but far from optimized. Many, many more trials would be needed to find the optimum PD controller

gains and configuration. It was not possible to do this as part of this project, but would be well worth the time taken. The drone's function library defines four different floats be sent each time the drone is to be moved. Having four axes of movement means that the drone is capable of performing rather sophisticated movements, rather than the jolted movements used in this project's tests that were often along only 1 axis at a time.

There will always be room for improvement in the fluidity and precision of a UAV's movements. This topic could nearly be a project all of its own.

8.2.4 Positional Awareness

The last major issue encountered during testing was the drifting of the A.R. Drone when not being guided by machine vision[1]. There is a short time in test HT2 where the drone must perform some timed commands without positional feedback, and its behaviour during this time was inconsistent and potentially very detrimental to the results of the test[2] This inconsistency was magnified in HT3, where the period of flight without positional feedback was much longer. This was one of the two reasons why the obstacle avoidance function never made it to testing.

It was slightly outside the scope of this project to develop a method to provide some form of positional feedback to the drone using objects other than the homing "beacon", but this topic has been studied in other projects, even here at USQ. Techniques exist, and are probably frequently used in official applications, that allow a UAV to take feature points from every incoming frame, and automatically track these points, providing a form of positional feedback. A technique such as this would allow the drone to "latch" onto another object while it is flying without line of sight to the target object.

Such a technique may take time to implement, but would be well worth its value in

[1] A basic test showing this was documented at the start of the "Homing Tests" chapter.

[2] The drone only completed a theoretical flight in this test anyway, so the actual error of the "blind flight" and its effects were undetermined.

programs such as those developed in this project.

8.3 Final Conclusion

Overall, the progress made in A.R. Drone automation as part of this dissertation could provide a great entry point for another individual to take on much more challenging tasks with the drone. Basic tracking and detection functions have been implemented and tested, and attempted to be woven into larger functions, such as homing tasks and obstacle avoidance. While these complex tasks did not reach a fully completed state, the role that these functions play and their interaction with other processes in a larger program was identified and properly applied.

Appendix A

Project Specification

Project Specification | USQ FOES ENG4111

Name: **Derek Long**

Topic: Development of an AR drone function allowing it to perform a simple homing task.

USQ Supervisor: Tobias Low

Co-Supervisor: Cheryl McCarthy

Enrolment: ENG4111, S1, 2013; ENG4112, S2, 2013

Project Aim: This project aims to improve the functionality of the AR drone by incorporating additional image processing functions in the drone. A homing function will be used as a testing function.

Programme: Issue A, 12th March 2013

1. Research possible solutions for poor battery life and flight time of the drone. (Additional activity that will support testing for the main project).
2. Perform study and write literature review focusing on current image processing functions existing in UAVs (tracking, homing, path awareness).
3. Study the drone's control systems, camera capability, and control device capability and use gained knowledge to develop several test functions implementing image processing. Functions will start at a simple level and will gradually apply more technique to implement more complex functions. A suggested order of functions is as follows:
    a. Have the drone find a coloured ball sitting on the ground, and hover over it.
    b. Sit the ball on an elevated position, adding elevation challenges to the function.
    c. Add obstacles to the drone's path, and develop the drone's capability to foresee and avoid these obstacles (possibly doors or windows).

    A proposed systematic approach to the tasks is given below:

    i. Acquire source code for the drone's controlling application (found on either android or iOS), to better understand the underlying functions and processes of the drone's control systems.
    ii. Analyse existing programs and code to identify best method of implementing solution.
    iii. Develop function within application using additional image processing libraries or resources as needed.
    iv. Test, evaluate solution and re-design if necessary.

As time permits:

Implement image processing functions on the drone to achieve more complex tasks.

Agreed

_____,	_____,	_____,
Derek Long	Tobias Low	Cheryl McCarthy

Appendix B

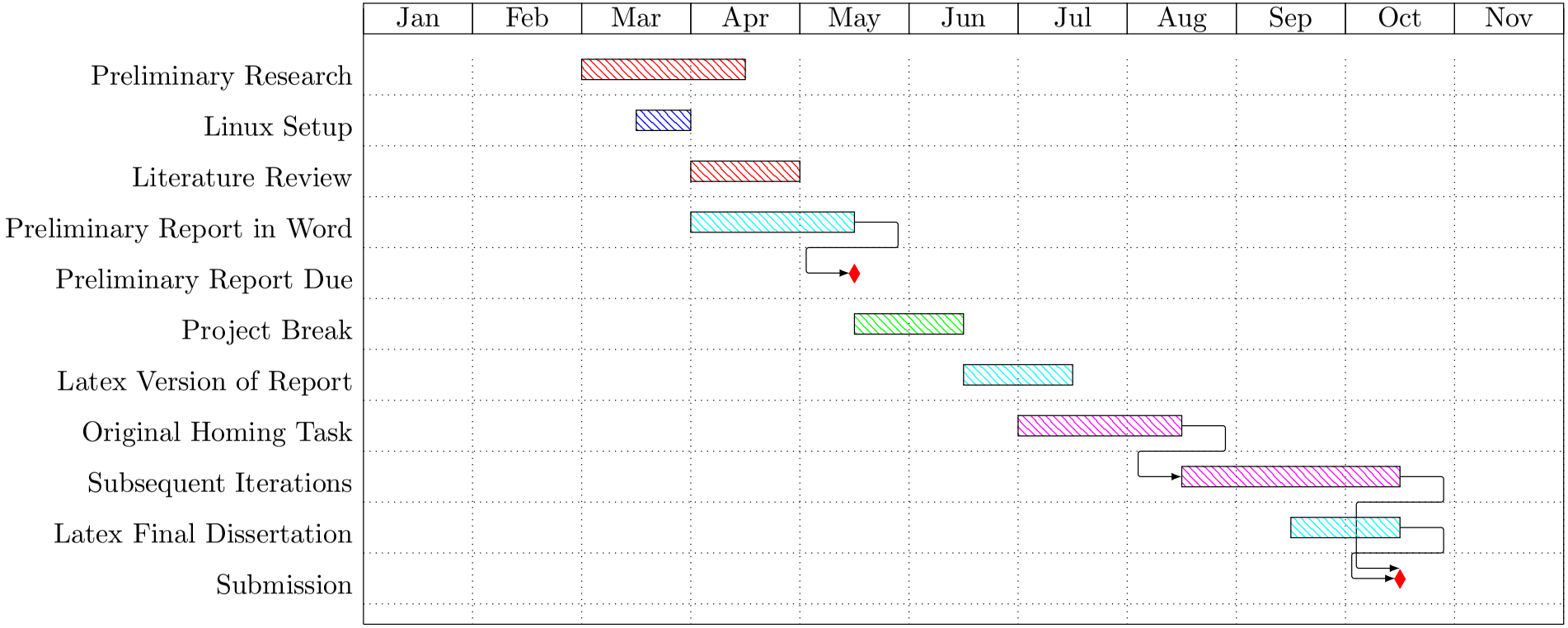
Project Timeline

The project timeline is on the next page.

The bars are coloured differently depending on the kind of activity being performed:

- Red - Research
- Blue - Setup
- Green - Break
- Teal - Report Writing
- Purple - Experimentation

2013 Project Timeline



CD/DVD Attachment

On this CD/DVD is all of the developed code used in this project's testing stage.

The directory for the test files is `ARDrone_SDK_2_0_1/Examples/Linux`. The entire ARDrone SDK has been included in the event that an assessor wishes to test the code personally. The built .exe files are in the "Build" folder located in the above mentioned directory.

Within the source code folder, the file `Video/display_stage.c` contains all of the code unique to that test. No other source or header files are heavily modifed (exceptions to this are the makefile, to specify the build target, and the control thread, to be turned off in experiments that don't use it).

References

Allili, M. & Ziou, D. (2006), Object contour tracking in videos by matching finite mixture models, *in* 'Video and Signal Based Surveillance, 2006. AVSS '06. IEEE International Conference on', pp. 35–35.

Baum, M., Faion, F. & Hanebeck, U. (2012), Tracking ground moving extended objects using rgbd data, *in* 'Multisensor Fusion and Integration for Intelligent Systems (MFI), 2012 IEEE Conference on', pp. 186–191.

Bellingham, J., Tillerson, M., Alighanbari, M. & How, J. (2002), Cooperative path planning for multiple uavs in dynamic and uncertain environments, *in* 'Decision and Control, 2002, Proceedings of the 41st IEEE Conference on', Vol. 3, pp. 2816–2822 vol.3.

Chung, J. & Yang, H. (1995), Fast and effective multiple moving targets tracking method for mobile robots, *in* 'Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on', Vol. 3, pp. V13–.

Deepthi, R. & Sankaraiah, S. (2011), Implementation of mobile platform using qt and opencv for image processing applications, *in* 'Open Systems (ICOS), 2011 IEEE Conference on', pp. 284–289.

Fernando, S. (2012), Object detection using colour seperation. Blogger, Mountain View, California, viewed 11th April 2013, <flqhttp://opencv-srf.blogspot.com.au/2010/09/object-detection-using-color-seperation.html>.

Feyrer, S. & Zell, A. (1999), Detection, tracking, and pursuit of humans with an autonomous mobile robot, *in* 'Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on', Vol. 2, pp. 864–869 vol.2.

Hong, C. S., Chun, S. M., Lee, J. & Hong, K. (1997), A vision-guided object tracking and prediction algorithm for soccer robots, *in* 'Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on', Vol. 1, pp. 346–351 vol.1.

Jilkov, V., Li, X. & DelBalzo, D. (2007), Best combination of multiple objectives for uav search & track path optimization, *in* 'Information Fusion, 2007 10th International Conference on', pp. 1–8.

Junkui, Y., Zaikang, Q., Fugui, L. & Tao, G. (2010), Study on uav flight based identification technology, *in* 'Control Conference (CCC), 2010 29th Chinese', pp. 1375–1380.

Khateeb, K., Awang, M. & Khalifa, O. (2009), Intelligent auto tracking in 3d space by image processing, *in* 'Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on', pp. 1744–1749.

Kim, J., Lee, C.-H., Young-Chul, Kwon, S. & Park, C.-H. (2011), Optical sensor-based object detection for autonomous robots, *in* 'Ubiquitous Robots and Ambient Intelligence (URAI), 2011 8th International Conference on', pp. 746–752.

Kout, P. (2012), Video streaming through opencv in linux. WordPress, viewed 4th April 2013,<http://petrkout.com/linux/parrot-ardrone-2-0-video-streaming-through-opencv-in-linux/>.

Lee, D., Lim, H. & Kim, H. (2011), Obstacle avoidance using image-based visual servoing integrated with nonlinear model predictive control, *in* 'Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on', pp. 5689–5694.

Lin, L. & Goodrich, M. (2009), Uav intelligent path planning for wilderness search and rescue, *in* 'Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on', pp. 709–714.

Luo, R. & Chen, T. M. (2000), 'Autonomous mobile target tracking system based on grey-fuzzy control algorithm', *Industrial Electronics, IEEE Transactions on* **47**(4), 920–931.

Majidi, B. & Bab-Hadiashar, A. (2005), Real time aerial natural image interpretation for autonomous ranger drone navigation, *in* 'Digital Image Computing: Techniques and Applications, 2005. DICTA '05. Proceedings 2005', pp. 65–65.

Mori, R. & Miyazaki, F. (2002), Gag (gaining angle of gaze) strategy for ball tracking and catching task, *in* 'Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on', Vol. 1, pp. 281–286 vol.1.

Park, S. & Lee, C. S. G. (1995), Very fast visual tracking algorithm using scanlines, *in* 'Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on', Vol. 3, pp. 2651–2656 vol.3.

Phang, S. K., Ong, J. J., Yeo, R., Chen, B. & Lee, T. (2010), Autonomous mini-uav for indoor flight with embedded on-board vision processing as navigation system, *in* 'Computational Technologies in Electrical and Electronics Engineering (SIBIRCON), 2010 IEEE Region 8 International Conference on', pp. 722–727.

Rasche, C., Stern, C., Kleinjohann, L. & Kleinjohann, B. (2011), A distributed multi-uav path planning approach for 3d environments, *in* 'Automation, Robotics and Applications (ICARA), 2011 5th International Conference on', pp. 7–12.

Seo, K.-H., Choi, T.-Y. & Lee, J.-J. (2004), Adaptive color snake model for real-time object tracking, *in* 'Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference on', Vol. 1, pp. 122–127 Vol.1.

Teuliere, C., Eck, L. & Marchand, E. (2011), Chasing a moving target from a flying uav, *in* 'Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on', pp. 4929–4934.

Zheng, B., Xu, X., Dai, Y. & Lu, Y. (2012), Object tracking algorithm based on combination of dynamic template matching and kalman filter, *in* 'Intelligent Human-

Machine Systems and Cybernetics (IHMSC), 2012 4th International Conference on', Vol. 2, pp. 136–139.