The University Of Sheffield.

Department of Automatic Control & Systems Engineering

Control of Satellite Dance

ACS330 MEng Group Design Project 2015/16

D4 : Final Report

Group Number: 1

Theodore Abbott - 120139831
Shiv Gangapersad - 130146238
Francesca Zoe Colaco Osorio - 130146179
Matthew Joy - 130146456
Stephen Allison - 100213528

Acknowledgements

We would like to thank Dr Jonathan Aitken for his invaluable contribution to this project, and Dr Bryn Jones, whose ACS 336 lecture series informed the construction of our Simulink model.

We wish to thank Overlander for supporting this project with a significant discount on component prices; Jason particularly who was kind enough to provide additional specifications for the selected components over those normally given.

We would also like to express our gratitude to Craig Bacon, Jack Powell and Anthony Whelpton for their continuous, friendly support and unyielding helpfulness.

Finally we must thank Prof. Sandor Veres & Dr Roderich Gross for their guidance throughout the academic year.

Abstract

This project focuses on the distributed control of a group of tabletop satellites through the use of vision-based computing techniques. The satellites are constructed and programmed to represent real satellite mechanics and formation flight in space, and they operate using reaction-based propulsion similar to the reaction control system (RCS) of a real spacecraft. They operate in a low friction environment consisting of a glass topped table and ball bearings to minimise rolling friction between the surface and the satellites. The aim of the project is to have the satellites perform a 'dance' semi-autonomously. This report assesses the various methods to implement computer vision in the satellite system, and covers a variety of designs that were proposed. Hardware consideration such as motor control, camera placement and landmark/grid design are discussed. This report also covers considerations about future work to be done on the satellite system.

Contents

List of Figures

1 Introduction

1.1 Project Background

Satellites are vital to a myriad of technologies from GPS to communications; they have become an essential part of the modern world. They are now integrated so deeply in our society that it is impossible not to use them. One of the biggest future challenges is that of transporting larger payloads into space; one possible solution is to launch several smaller rockets which are then docked in order to distribute the payload, as was achieved with the international space station (Mraz 2001).

The use of multiple satellites simultaneously requires more efficient & safer methods of control, such as formation flying of satellite groups. Additionally organising flight trajectories in progressively more crowded space during formation flying necessitates object avoidance; optimisations are discussed in (Seong and Kim 2013). The ability to organise formation flight paths is therefore critical to the furtherance of space technology.

Several space missions now rely on distributed control and formation flying of clusters. This is much more viable in terms of failure, as if one satellite fails in a cluster, the rest of the group can still carry out limited or even full performance, whereas failure of a monotonic satellite system is virtually impossible to fix (Sabol, Burns, and McLaughlin 2001).

Formation flying adds a lot of flexibility to space missions and it greatly reduces cost and complexity as opposed to singular larger missions, where a considerably larger amount of fuel is used owing to mass payload ratios (Cui, Li, and Gao 2006).

Formation flying of satellite clusters has proved especially useful in the execution of missions that agencies like *NASA* deemed almost impossible to be carried out with larger singular units. Examples would be the efficient study of the Earth's magnetosphere, or even the Orion Program designed to be a low-cost demonstration of GPS (Sabol, Burns, and McLaughlin 2001).

1.2 Aims

This project seeks to develop three table top model satellites that may be used to demonstrate methods of formation flying and obstacle avoidance. To best resemble an orbit in space the satellite-table interface must be extremely low friction. The satellites must manoeuvre using some form of fluid based propulsion to effectively mimic the propellant based reaction control systems (RCS). Despite research into alternative forms of movement such as electromagnetic formation flight as in (Kwon 2010) RCS is still the primary form of actuation for orbiting satellites (Mu and Zhang 2014). After completing construction, the satellites will be programed to perform formation movement in five degrees of freedom (DoF): X and Y planer movement across the table, roll and pitch about the centre point of the satellite frame, yaw in the axial direction about the satellite stand stem. This is shown in figure 1.

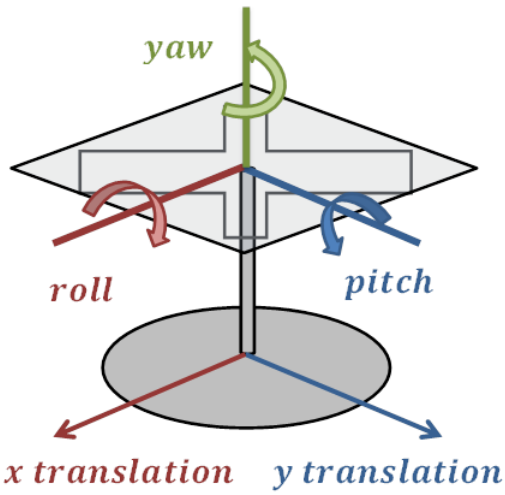

Figure 1: Diagram demonstrating the 5 DOF of the satellite

1.3 Objectives

1.3.1 Hardware Objectives

- Deduce required performance of the hardware required to meet the specification of the requirements document.
- Select appropriate hardware for the satellites taking into account cost, reliability and compatibility.
- Negotiate with suppliers to get the best possible deal on components and place order.
- Construct the satellites to specification.
- Write the low level code to control the brushless electronic speed controllers (ESC) and have them track a reference speed.
- Make a control/test remote for the demonstration.

1.3.2 Model Objectives

- Create a model of the system in the 3 Degrees of Freedom configuration, (DOF)
- Derive a suitable control algorithm for implementation on the real life 3 DOF system
- Extend the model to 5 DOF, allowing for simulation of possible hardware configurations and control strategies

1.3.3 Computer Vision Objectives

- Develop computer vision algorithms and software architecture, and implement them
- Combine computer vision and localisation into working system

1.3.4 Autonomy & Localisation Objectives

- Identify a method for satellite localisation
- Select a method for satellite formation movement and implement it onto the satellites

1.3.5 General

- Integrate the ODROID with the arduino, camera module and desktop.
- Produce complete documentation to enable future groups to easily use this project as a platform for future research and development.

1.4 Report Structure

This report begins with a description of the hardware redesign, covering both specification and selection of replacement components. Additionally the hardware architecture and testing is included along with a description of the low level code used by the onboard microcontroller to control the components and feed the necessary information to the onboard computer.

The report then address the design of a model of the system; first deriving the satellite dynamics, then presenting the resulting Simulink model implementation, simulation results and repercussions for the control of the real system.

The report discusses the different techniques for satellite localisation, are then examined by looking at self localisation compared to identifying the position of foreign objects.

Penultimately the report looks at the formation movement of the satellites, this is done by looking at various different control methods dictating how the satellites would move relative to one another.

Finally conclusions and a suggestion of future actions are made.

2 Hardware Performance and Selection

2.1 Requirements

The requirements that the components should meet are given in Appendix A; these are a high level description of the satellite motion. A hardware design was developed to meet these requirements, and specified the performance needed from individual components. The chassis for the satellites were completed before the start of the project; hardware was therefore selected to be compatible with these frames. This resulted in certain constraints on the size and weight of components as well as the power of the propulsion systems.

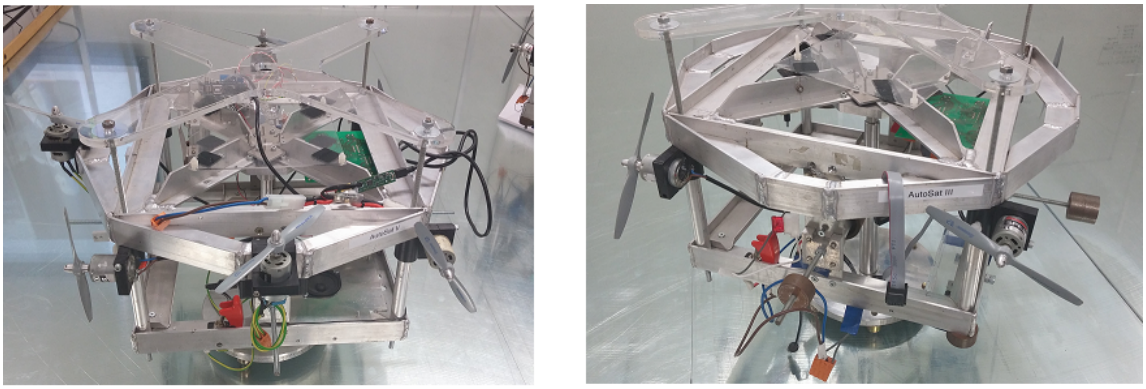

Figure 2: (a) AutoSat II, (b) AutoSat III

The satellites will be used for further research projects, so the hardware must have good documentation and support in order to be quickly comprehensible for future users. The hardware inherited at the beginning of the project had no documentation and was largely mismatched; It was therefore decided to redesign the satellites keeping only the chassis. Figure 2 shows two of the inherited model satellites in addition to the glass surface of the table.

2.2 Actuation

The required mass and performance of the satellites is specified in the requirements, it therefore made sense to start with the required propulsion system and then select the rest of the components to be compatible.

2.2.1 Actuator Placement

Requirement 2.3 specifies that 8 independently actuated propellers must be used to represent satellite RCS, the placement of these actuators will have a large effect on the performance of the satellites. There are two main placement options to achieve 5 degrees of freedom (Requirement 1.1).

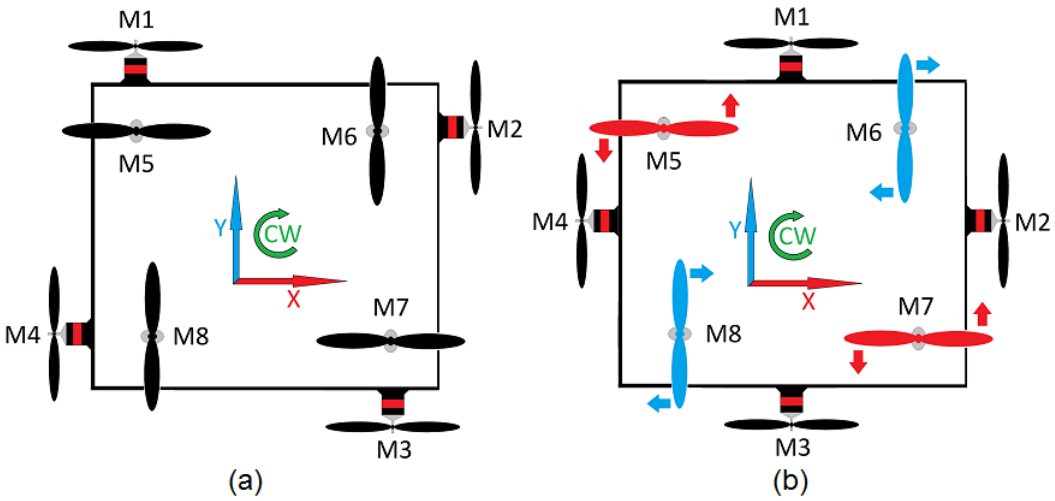

Figure 3: (a) Motor configuration 1, (b) Motor configuration 2

Figure 3 shows option 1; this requires that motors M1-M4 are reversible during operation. To move in the positive Y direction M1 must run forwards while M3 runs in reverse, the pairing of opposite rotating motors in needed to move in all directions on the X Y plane. Motors M2 and M4 may also be activated in reverse along with M3 to make up for inefficiencies of propellers running in the opposite direction to their design. To yaw counter clockwise (CCW) at least 2 motors must be run in reverse.

Figure 3 shows option 2; this does not require motors be reversible, however it does require that motors M5 and M7 run in the CCW direction to yaw the satellite using torque generated (To yaw the satellite clockwise (CW) M5 and M7 would be run at the same time). It is worth noting that there are propellers designed to run in the CCW direction, it is running in the opposite direction to the design direction that introduces inefficiency.

One significant disadvantage of option 1 is that well designed propellers running in reverse are highly inefficient and produce much less thrust than in the normal direction, this would require that motor operating in the forward direction reduce thrust to the level of the reversing motor(s). The option selected will affect the motor and electronic speed controller (ESC) selected: for option 1 the speed controller must be able to reverse direction during operation. Option 2's reliance on torque roll to yaw is weaker than the yawing of option 1 because of the lack of direct actuation, instead relying on the secondary characteristic of torque roll. The actuator placement will not be decided until thrust and torque with reversing propellers can be measured as it is difficult to predict accurately without advanced computational modelling.

2.3 Propulsion

In order for the satellites to respond quickly to large scale disturbances (Requirements 9.2) the motors must be able to rapidly change their thrust and therefore speed. This requires

that the angular momentum for any given thrust be minimized. This can be achieved either by reducing the moment of inertia of the rotor-propeller combination or by having it spin more slowly. This is a problem that is very common with fixed pitch multirotors and is typically overcome by using slow spinning higher torque motors with larger propellers.

The model satellites will never be moving through the air at any great speed (Requirement 3.1); therefore a large propeller spinning slowly will produce more force than a smaller, higher velocity propeller which would be able to continue producing thrust when travelling through the air at greater velocities (McCurdy 1985) (Gordon 2016). This is similar to how fast aeroplanes tend to use fast spinning jet engines compared to slower planes using a slower but larger propeller. Another consideration is safety; the energy of a rotating body is proportional to the square of its frequency, additionally the tip speed increases linearly with increased blade length, however thrust increases with a quadratic relationship allowing for increased thrust per tip speed. Larger slower propellers are therefore safer for any given thrust during both catastrophic failure and tip collision.

The satellites are expected to weigh 8.5kg ($\pm$20%) (Requirement 2.10), and be able to achieve a speed of 5cm/s (Requirement 3.1). Assuming the maximum weight of 10.2kg and the top speed should be achieved in 1 second the minimum required thrust can be found as shown in Equation 1, which describes the minimum theoretical thrust to meet performance requirements. Note this is a theoretical minimum, containing many assumptions such as negligible rolling friction and air resistance; as such the actual thrust required will be larger.

$$F = ma = m\frac{\delta v}{\delta t} = 10.2\frac{0.05}{1} = 0.51N \tag{1}$$

Most datasheets do not directly state the thrust produced by their motors, however it can be approximated first by calculating the power draw of the motor and then by using Equation 2, corrected equation for propeller static thrust taken from (Staples 2016). The constants included in the equation come from the fluid properties of air at sea level and typical propeller losses. An important factor in brushless motor selection is the rotations per minute per volt (Kv); motors with the same power output but a lower Kv value have more torque and are better able to accelerate larger propellers. Note: Load RPM is typically between 0.5 to 0.75 multiplied by the Kv rating of the motor for a well matched propeller motor combination (Stabler 2016).

$$F = 1.225\frac{\pi(0.0254 * diameter)^2}{4}(RPM * 0.0254 * pitch * \frac{1minute}{60seconds})^2(\frac{diameter}{3.29546 * pitch})^{3/2} \tag{2}$$

Lastly the degree of linearity of the propulsion system must be considered; where fine control is needed a high degree of linearity about the operating point is advantageous as it allows for

simpler control and less error when linearised. As discussed the selected motors will be more powerful than the selected propeller is designed for to allow for fast thrust changes. This excess of thrust at high speeds means the propellers will normally operate slowly; propellers which are linear at the bottom of the rotational velocity thrust curve are therefore desirable. Brandt (Brandt and Selig 2011) performed thrust and torque tests for many propellers in the 9 to 11 inch range, which corresponds to a low Reynolds number due to scaling. Reynolds number is "a dimensionless number used in fluid mechanics to indicate whether fluid flow past a body steady or turbulent" (Stokes 1851). It is frequently used in fluid dynamics to describe how the scale of objects affects the behavior of fluid flow around it. From Brandt's data set the data sets it can be seen that propellers with wide outer blades tend to perform in a more linear fashion at lower shaft velocities but more quickly tale off in performance at higher shaft velocity. It is therefore reasonable to prefer propellers with wider blade tip sections for this project as low thrust linearity is preferable and high thrust is not required.

2.3.1 Propellers

The primary characteristics of interest for hobbyist propellers are:

- Diameter; simply the tip to tip length of the propeller. The limiting factors on size are motor selection, physically fitting the disk swept out by the propeller on the satellite frame and vibration and resonance avoidance which is especially important for this project due to the camera and gyroscopes being greatly affected by vibration. The diameter is typically the first measurement in the propeller specification and is normally given in inches, e.g. a 10x4.7 propeller is 10 inches long.

- Pitch: Describes the angle of the propeller; higher pitch propellers move larger volumes of air with each rotation allowing the same thrust to be achieved at lower speed. Pitch is defined as, 'the distance a propeller would move if rotated once through a solid,' adapted from (Rudow 2016). The pitch is typically the second measurement in the propeller specification, e.g. a 10x4.7 propeller has a pitch of 4.7 inches. A diagram showing differences of pitch in a propeller is shown in figure 4.

- Optimized speed which describes the rotational velocity at which the propeller performs best, slow fly (SF) propellers are preferred for their better performance at low shaft speeds corresponding to low rotational velocity and wide blade tips indicating a more linear thrust shaft speed relationship at lower shaft speeds (ibid.).

Larger, slower spinning propellers are preferred for this project, therefore the selection criteria for propellers are large size, high pitch and slow fly optimised.

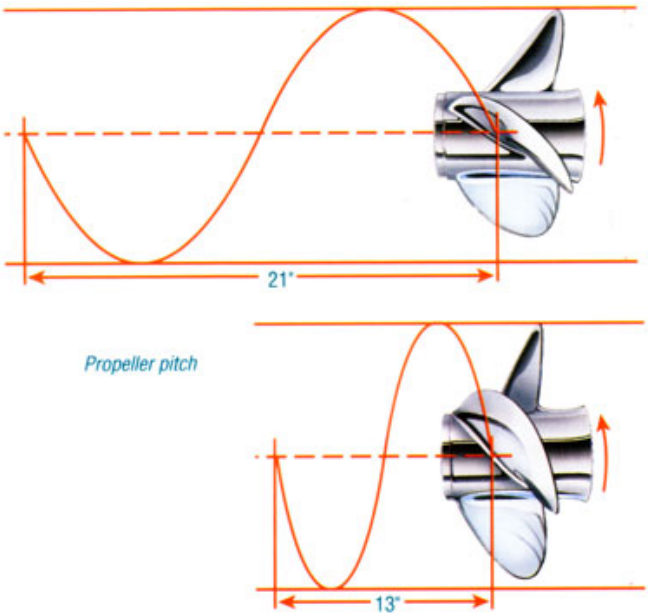

Figure 4: Diagram showing propellers with high (top) and low (bottom) pitch (Rudow 2016).

2.4 Sensing

The satellites are required to self-localise to within 3cm (Requirement 1.6.3) for the duration of a 5 minute demonstration (Requirement 6.1). Self-localisation will be achieved by the use of an upward facing camera attached to the top of each satellite. A grid will be hung above the operating environment with many recognisable shapes; the shapes will be read by the camera and compared to a lookup table of known shapes and their positions. Further detail on self-localisation is outside the scope of this report. The satellites will be required to angle themselves on their tilt bases (Requirement 1.3.2); orientation sensing will be required for this.

3-axis gyroscopes (Requirement 2.5) may be used to sense yaw, pitch and roll. These gyroscopes must be compatible with the Arduino Uno microcontrollers that were preselected for this project (Requirement 2.7). The gyroscope must be accurate and have a low latency of readings to allow for rapid correction; fortunately most hobby gyroscopes are orders of magnitude faster and more accurate than is required.

2.5 Controller

Each satellite will be equipped with an Arduino UNO R3 microcontroller and an ODROID XU3 for computationally intensive tasks (Requirements 2.7 & 2.8). Speed controllers and sensors must be able to interface with these computers. Requirement 8.5.2 states there must be some form of low voltage protection for the batteries.

2.6 Hardware Selection and Supply

Components were selected to meet not only the performance requirements detailed above, but also budgeting restraints and availability in the numbers needed. This was an iterative process as once a component was found that seemed satisfactory the supplier was contacted to negotiate a discount. Lastly the hardware was ordered.

2.6.1 Propellers

Tower pro 10x4.7 slow fly propellers were selected, the primary reasons for this were:

- Low price and availability in bulk, this allows for the breakage of some propellers during testing.
- Designed specifically for low speed applications, particularly on multirotor aircraft.
- Largest size available to fit option 1 on the satellite.
- High Pitch.
- Well balanced from the factory and easily balanceable on a suitable rig avoiding the problems of vibration and resonance.

2.6.2 Motors

Motors were shortlisted based on having a low Kv rating with high power as this translates to the high torque required for larger, slow spinning propellers. If available the numbers of windings were also compared, as these give another indication of torque. Secondly they were compared on their compatibility with 'Slow Fly' Propellers which are specifically designed for low RPM use cases. The Overlander Tornado Thumper V3 3536/08 1050KV brushless out runner motor was selected because of its compatibility, and because the supplier was willing to provide a large discount due to the number being ordered. The resolution of the motors, in terms of the smallest speed change they can achieve is limited by the ESC and ESC controller resolution as will be discussed later.

$$\begin{gathered} F = 1.225\frac{\pi(0.0254 * 10inches)^2}{4}(0.5 * 1050K_v * 11.1V * 0.0254* \\ 4.7inches * \frac{1}{60})^2(\frac{10inches}{3.29546 * 4.7inches})^{3/2} \\ F = 4.33 \end{gathered} \tag{3}$$

Calculating the estimated static thrust of the Tornado Thumper V3 3536/08 1050KV at 0.5 Kv rating using equation 2 yields a predicted thrust range at max throttle of 4.33N to 9.74N (Equation 3). It therefore far exceeds the minimum theoretical thrust requirement; this excess of thrust will allow the motors to spin slower offering lower angular momentum

and reflecting the high torque of this motor. Slightly cheaper motors were available, however these were from less reputable brands and used lower grade components and as discussed above, reliability, longevity and documentation were important for this project: the small extra cost was therefore acceptable. This motor requires three phase alternating current to run; an ESC was therefore required for each motor to run at independence speeds.

2.6.3 ESCs

The selected ESC must provide sufficient current at the correct voltage (11.1v) for the motor during operation, ESC's typically have two current ratings; constant (the maximum current that can be constantly supplied) and burst (the maximum current that can be applied for a short time, typically 30 seconds). A 30A ESC is recommended for the selected motor, however, due to the nature of the model satellites, no motor will ever run at maximum power for more than a few seconds. As shown above by the excess thrust, this motor will likely never operate at more than 50% power. For safety only ESCs able to handle 30A burst current were selected, a lower constant current rating was acceptable as the satellites would be moving dangerously fast after even having just one motor running at 100% for more than a few seconds.

The resolution of the speed controller is also important for the control problem; insufficient resolution would result in a loss of granularity of control and may even cause instability. Given that the motors will usually operate below 40% thrust due to their high power a high resolution of control is needed as there must be a sufficient number of steps in power between 0% and 40% thrust rather than over the whole thrust range. However the servo control library for the Arduino that will be used to control the ESC's only accepts integers in the range from 0 to 180 limiting the resolution. Almost all none specialist hobby ESCs have a 8bit resolution giving 256 discrete speed steps, more than enough to exceed the 180 step limit set by the Arduino.

Speed controllers to meet these specifications normally cost in excess of £10, a considerable cost per satellite as 8 would be needed. The popularity of model multirotors has resulted in an alternative option; a 4IN1 ESC is a single component containing 4 independent ESCs, significantly reducing cost and complexity as only 2 are needed per satellite. The Overlander XP2 QUAD 20A was selected because of its 40A bust current rating, clean wiring and inbuilt low voltage shutoff and alarm eliminating the need for a separate battery alarm. Additionally this ESC will provide the 5V power supply needed for the ODROID and Arduino.

2.6.4 Batteries

Lithium polymer (LiPo) batteries were selected for their high energy density. The three specifications of interest with LiPo batteries are voltage capacity and discharge rating typically written as 'C rating'. The capacity expresses the total energy output of the battery expressed in Amp hours at the batteries voltage as shown in Equation 4.

$$Energy = Current * Voltage * Time$$

(4)

The C rating expresses the safe discharge rate of the battery in terms of the batteries capacity Equation 5. For example a battery with a 2Ah capacity and a 10C rating has a maximum discharge current of 20A which for an 11.1V battery is 222W. Some batteries also come with a maximum burst C rating.

$$Crating = \frac{DischargeCurrent}{Capacity}$$

(5)

The capacity and C rating of the needed battery can be calculated from the selected motors and desired runtime as follows. The maximum current draw of each motor is 30A and the desired runtime is 5 minutes (Equation 6).

$$Capacity = Current * Time = 30 * \frac{5}{60} = 2.5Ah$$

(6)

To run one motor at full speed for 5 minutes a 2.5Ah battery is required. Motors will very rarely operate at 100% load, and often less than half of the motors will operate at the same time. As such it will be assumed that 4 of the 8 motors will be running at 40% power on average. This estimate is acceptable because a battery running out of charge is not a safety critical problem and as discussed in motor selection the motors will likely never exceed 50% power. (Equation 7)

$$Capacity = 4 * 30 * 0.4 * \frac{5}{60} = 4Ah$$

(7)

This assumption yields 3Ah required capacity. Next the C rating of the battery must be calculated; the worst case scenario must be considered as rapid discharge of LiPo batteries will often cause damage and potentially even fires. It will be assumed that 6 of the 8 motors will be running at 100% burst current load of 30A each, 6 motors running at 100% is the worst case as no more than two of motors M1-M4 should run at the same time at full load as they are in opposition. (Equation 8)

$$Crating = \frac{6 * 30}{4} = 45$$

(8)

One final consideration is the use of multiple batteries per satellite, this has some advantages:

- Continuous testing as batteries can be alternated between charging and use while working on only 4 motors at a time. Both batteries can then be used in parallel for the 5 minute demonstration or when all 8 motors are being tested.
- Safer operation as each battery contains less energy so shorting will present lower risk.
- Redundancy, the loss of one battery still allows a satellite to be operated if the motor usage is kept low.

In accordance with these specifications two Overlander 2200mAh 3S 11.1V 30C LiPo Batteries per satellite were selected for their 50C burst current rating and high quality construction. The continued selection of Overlander parts also allowed a larger discount to be negotiated.

The very large power density of LiPo batteries requires a range of safety considerations and actions, such as fusing the batteries, these will be discussed later.

2.6.5 Gyroscope

Many off the shelf gyroscopes merely output the direct reading, thus much data processing must be done to extract useful knowledge about orientation. Continuing with the importance of documentation and compatibility the Arduino 9 Axes Sensor Shield was selected as it directly outputs orientation from a 3-axis gyro and does not require further signal processing. It has a polling rate of ± 2000 degrees per second and a 16-bit resolution. It is designed specifically for use with the Arduino Uno R3 and is open source with good documentation. This board also comes with a 3-axis accelerometer and magnetometer which may augment the gyroscope and camera data for orientation and self-localisation.

2.7 Ordering

A list of components of interest is given in Appendix K. A significant discount of nearly 40% was negotiated with Overlander due to the order size and academic nature of this project. As discussed with the project supervisor, additional money was supplied to the original £500 budget from research funds, the total cost of components for the three satellites was £753.51.

3 Electronics Architecture

Each line or arrow represents a physical wire.

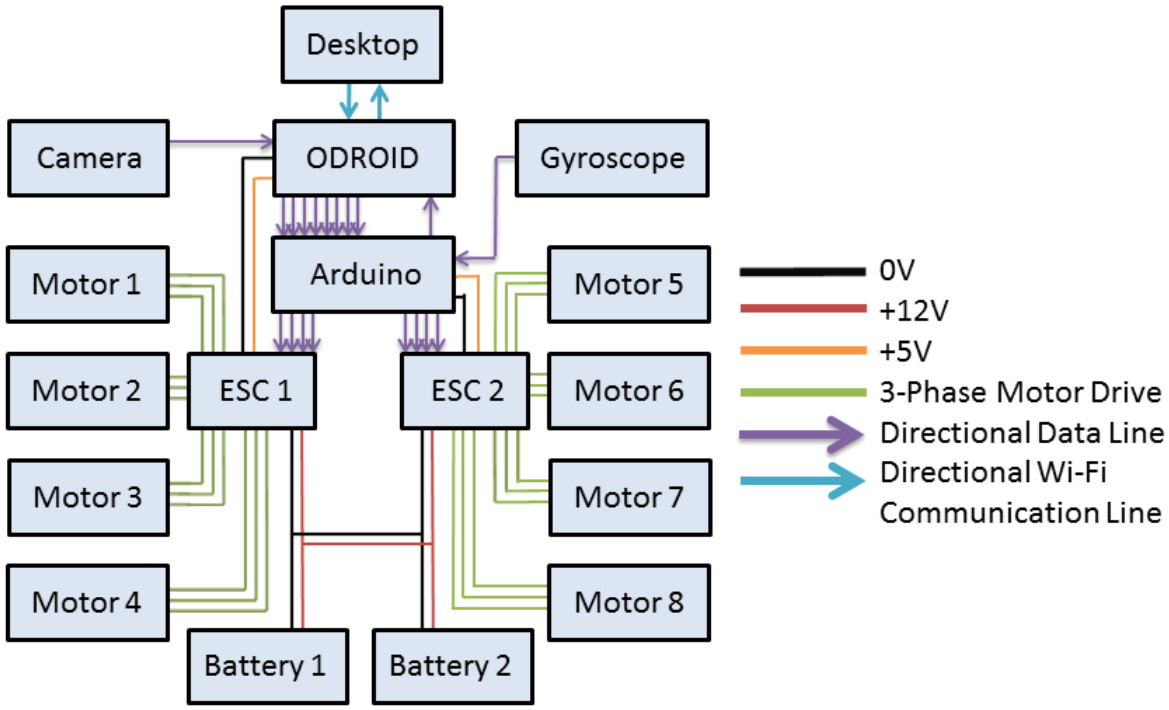

Figure 5: Wiring Diagram

The ODROID is the main controller of the satellite, communicating with the desktop to receive program start commands and send live actuation and sensing data, it is also responsible for all of the image processing and decision making of the satellite. It has an input from the camera which it analyses for self-localization and a serial input from the Arduino which gives 9 axis orientation, acceleration and magnetometer data. The ODROID also outputs 8 analogue signals between 0v and 5v vaults corresponding to the desired thrust from each motor.

The Arduino is responsible for processing the gyroscope, accelerometer and magnetometer input from the gyroscope board, processing it to be simple and useful orientation and acceleration data before passing it to the ODROID. When the satellite first receives power, the Arduino initializes and tests the ESCs before allowing them to complete their setup process. During operation it processes the analogue thrust demands from the ODROID to negate the effects of noise, before generating the appropriate PWM signal for each of the 8 channels of the ESCs.

Each ESC receives power from both batteries to prevent one battery from being depleted before another if there is an uneven load between the two ESCs. One ESC powers the Arduino

while the other powers the ODROID as they both have 5v 3A DC power supply built in. Each ESC also provides the correct 3-phase switching sequence, frequency and amplitude to power the motors.

The low voltage alarms have been omitted from the figure for the sake of simplicity. These alarms plug into a separate balancing connector on the battery and sound when they detect that any cell in the battery and within 10% of the safe low voltage.

4 Power System Testing

Once the batteries arrived they were fused with 60 amp fuses to protect against errors during the initial wiring. Once the wiring was complete and had been tested the fuses were upgraded to handle the full 110 amp burst current that the batteries are capable of.

The thrust characteristics of the motors were tested with the selected propellers using the Turnigy thrust measuring setup shown in figure 6.

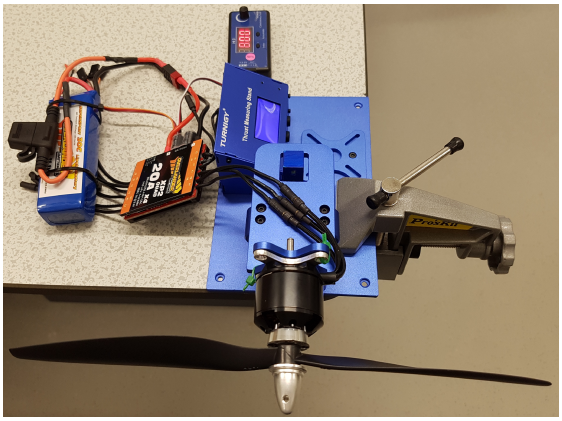

Figure 6: Turnigy Thrust Measuring Setup

Three measurements were taken for each power request (PR) and the results were averaged. Where a difference of more that 50g was found between measurements at the same PR, a further 3 readings were taken and averaged. The averaged measurements are given in Appendix L and plotted in figure 8. It can be seen that the thrust rises steadily to a peak of 1160g (11.3N) at PR of around 45% before leveling off, this fits well with the upper limit power estimations (9.74N) that were done during ESC and battery selection stage. This leveling off is due to the motor propeller combination. As discussed above these motors are very powerful for the actual thrust requirements due to the need for them to be able to change their speed very quickly, this means that when combined with the slow fly optimized propellers maximum thrust us achieved at only around 45% PR. The slow fly propellers are designed to produce their maximum thrust at around half the angular velocity that the motor propeller combination is actually capable of. This thrust rotational velocity relationship

is typical of the propellers described in Mc Cormick (McCormick 1979) as summarized by Spakovszky (Spakovszky 2016). The PR to thrust relationship remains very linear up to around 35% PR or 90% of maximum thrust.

The thrust characteristics were then tested with the propellers rotating in reverse, the same test method was used as above: the results of this test are showing in figure 8, while the raw data is given in Appendix K. It can be seen that the peak thrust output is only 29.3% of the peak thrust in the forward direction. More disturbingly the PR – thrust relationship is far less linear when the propeller is reversing and there is no clear region where the relationship could be well linearized. Nevertheless an attempt at linearisation has been undertaken as discussed later, however the error at 10% thrust is 50% and at 20% PR is 40%Ȧdditionally when running in reverse the average thrust over the operating range of the motors is only 15.6% of running in the forward direction. These results clearly show that this motor propeller combination is not suitable for running in reverse, the large efficiency loss combining with the non-linearity make this option overly challenging for substandard performance. As such a motor configuration that does not require propellers to run in reverse should be selected. While other projects have been successful with propellers running in reverse, these were not slow fly optimized propellers. Projects requiring reversing propellers demand careful propeller selection, which should be made along with extensive testing due to the difficulty of simulating propeller behavior. Due to the time and budgetary constraints of this project, extensive testing with multiple propellers was not possible.

The torque roll produced by the motor was more difficult to quantify as no rig to test it was available. As such the angular acceleration was calculated by timing the time taken for the satellite to complete a rotation under its own power from an initial rotational velocity of 0rads/s resulting in a time of 3.0 seconds. This was combined with the known axial rotational inertia of $0.0875kgm^2$. This was estimated using standard inertial models to find the torque using the standard constant acceleration equations and torque inertial relationships given in equations 9 and 10. This method had the advantage of accounting for the stiction and rolling friction of the bearings as the calculated torque is the remaining torque after overcoming stiction and rolling friction better representing the behavior of the satellite.

$$\theta = \theta_0 + \omega_0 * t + \frac{1}{2} * \alpha * t^2 \tag{9}$$

$$\tau = I * \alpha \tag{10}$$

This resulted in a combined axial torque of the two motors running at 50% of 1.5Nm.// This power layout provided far more thrust than was required to move the satellites at the required speed, however, this excess of power allows the motors to very quickly change their speed and go from 0-100% thrust in less than 200ms. This very fast response time allows movement, particularly in 5 DoF to have very fine control and rapidly act against the effects

of disturbances.

The resolution in terms of the number of steps of motor speed over the operating range is therefore 72 corresponding to a minimum thrust step size of approximately 16.1g (0.157N) allowing the satellite to alter its acceleration in step sizes of 0.0146 meters per second squared.

Due to the lack of reversibility of the speed controllers and the inefficiency of running directional motors in reverse motor configuration 2 was selected. This configuration has the additional advantage of decoupling the top motors from the horizontal motors when the satellite is operating in 3 DoF.

5 Arduino Code and Safety

The Arduino acts as a high level motor and sensor controller, passing data between the ODROID and the ESCs. The programming language is a modified version of C and is written and complied within the Arduino IDE.

When controlling the motors the Arduino takes 6 inputs between 0V and 5V on its analogue pins; a dead zone is then applied to these inputs as a safety feature to counter motors being turned on by noise in the readings. This analogue voltage is read by a 10 bit analogue to digital converter and then mapped to the 0-180 range of the servo control function. This function generated a PWM signal with a duty cycle of between 10% and 20% which is the standard expected range for hobby electronics. The PWM capable pins on the Arduino are run off 3 separate timers so run at different frequencies, this frequency discrepancy means each motor behaves differently depending on which pin it is attached to. Some experimentation was done to get around this issue such as creating a program that adjusts the PWM signal for each pin depending on the clock rate. However a simpler solution was found, by using an Arduino library found online (Arduino 2016) the PWM frequency in the Arduino could be fixed, for the purposes of this project a 300Hz frequency was selected as this is the upper end of what the ESC's can reliably read and provides the shortest delay between control signals.

Demo and test mode unit: In order to demonstrate and test the hardware a wired remote control unit was designed. This unit has 8 potentiometers allowing each motor to individually be controlled with a high degree of granularity; additionally a dead man's switch was implemented as discussed below. This unit plugs into the 6 analogue input pins and one of the digital pins in order to allow the satellite to be controlled without any modifications to the code on the Arduino.

Dead man's hand switch: One of the critical safety features of the hardware is the ability to shut off the motors immediately and with 100% reliability should anything go wrong. To

this end the hardware was designed with a momentary switch which must remain depressed by a user at all times when the satellite is working, on the same clock cycle as the switch is released the power to the motors is cut for 5 seconds giving sufficient time to disconnect the battery if there is a problem. To achieve this, a Boolean value which defaults to zero at the start of every cycle is used, the status of the switch is checked and if the switch is depressed the Boolean value is set to 1 for the duration of the cycle. This means the motors cannot keep spinning unless the switch is depressed, even if the Arduino encounters an error as the ESCs immediately activate motor breaking if they do not receive the required PWM signal at the agreed frequency.

In order to prevent the satellites from colliding with each other a protective bumper base was constructed, this bumper attaches to the base of the satellite and prevented anything from getting within 10cm of the outermost extremities of the satellites.

Figure 7 shows the layout of the components in the final satellite prototype. Not the 3D printed camera mount used to elevate the camera over the ESC to prevent heat buildup. 3D Printing was selected for prototyping due to the fast turn around time allowing for rapid fitting.

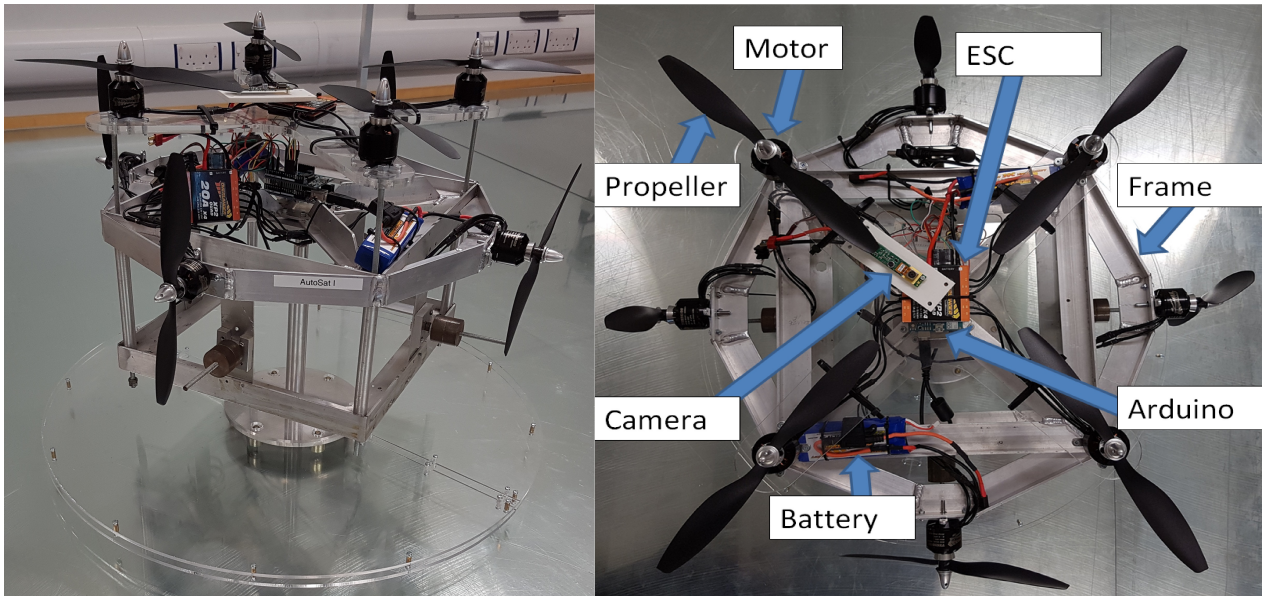

Figure 7: AUTOSAT1 showing the protective bumper base (left) and component placement (right).

6 Modelling & Simulation

6.1 Motivation

The aim of the model was to enable the development of a suitable control algorithm without implementing it blind on the hardware, reducing the risk of costly and potentially dangerous collision and propeller destruction had the satellites been tested without a full understanding of their behaviour.

6.2 Fan Assembly Behaviour

It is necessary to estimate thrust, T and torque, Q produced by each motor and propeller pair given the Pulse Width Modulation, (PWM) duty cycle across it. The motors were chosen to exhibit a linear PWM-thrust relationship around the operating point. This allows effects such as inductance to be assumed negligible, leading to the assumption that the thrust output of the motor is linearly proportional to the PWM voltage applied to it. The static response of the motors was determined using the experimental approach described in section 4. A relationship was then fitted in MATLAB using a linear regression method accessed via the fit.lm command, over the operating range 7.5 - 35 PR as seen in figure 8. The value of the torque produced by the motors at 50% of full power was estimated by observing the time taken for the satellites to complete one full revolution from standstill: assuming constant acceleration allowed the estimation of the acceleration produced by the motor torque. As both torque and thrust are related to the rotational velocity of the motor, the torque to power demand relationship was assumed to have the same shape as the thrust to PR relationship, but scaled by the value of torque at 50 % divided by the value of thrust at 50% .

The dynamic motor response could be approximated by a 1st order transfer function, thus accounting for the time taken for the motor to reach the required velocity: however the response time of the motors is so quick that in this application the dynamic response can be assumed to be instantaneous.

There are many additional aerodynamic and gyroscopic effects associated with any rotor, such as blade flapping and induced drag.(Mahony, Kumar, and Corke 2012) These are disregarded in many of the quadcopter models researched, and due to the lower angular velocities and speeds at which the satellite motors will operate, these effects can confidently be ignored.(Fernando et al. 2013)(Jirinec 2011).

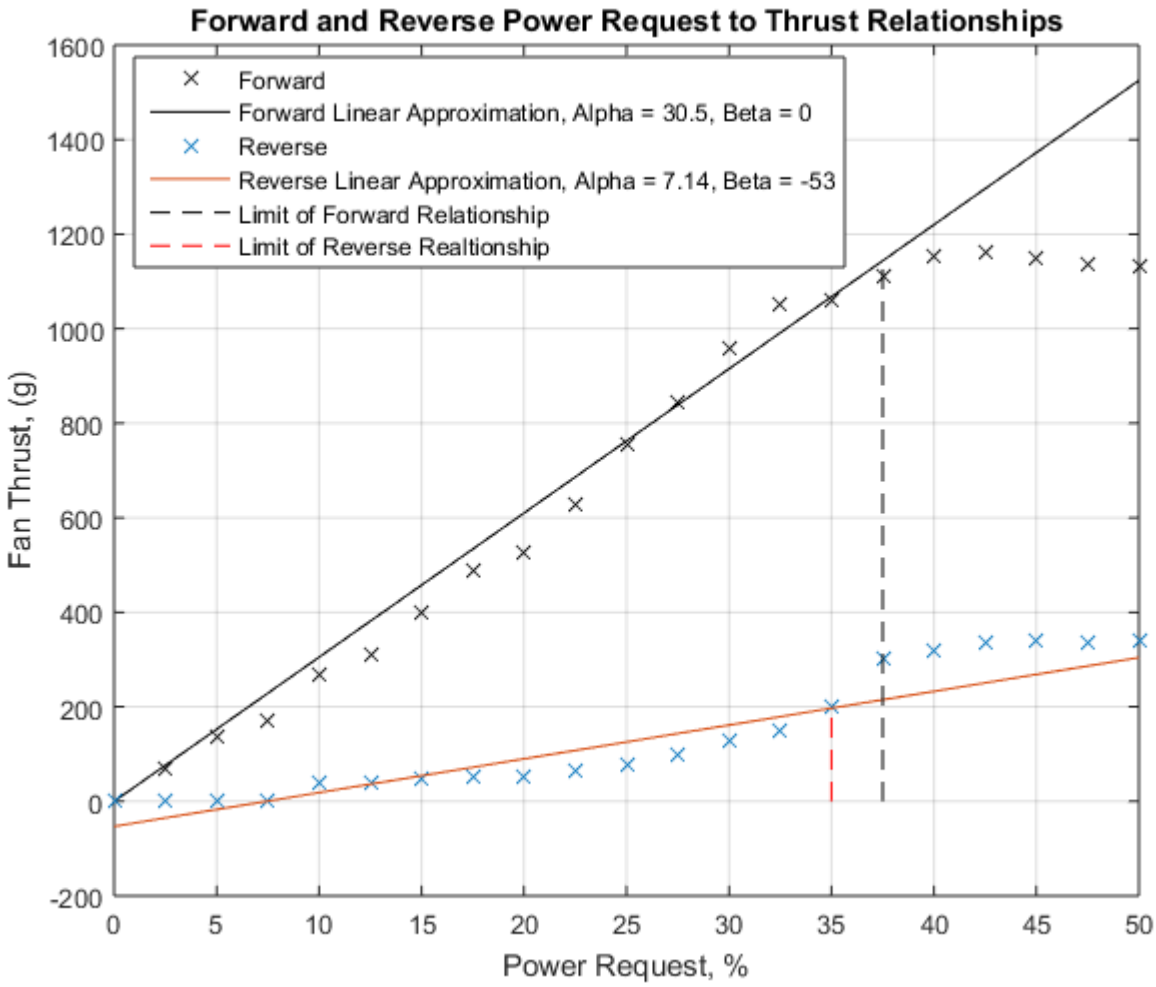

Figure 8: Graph showing the linear relationships fitted to the forward and reverse motor thrust data tests.

6.3 Rigid Body Dynamics

Although structurally different from the typical quadcopter configuration, the satellites exhibit similar dynamics, having the same motor-propeller actuation, x,y position and roll, pitch and yaw. Quadcopter simulation and control is a standard and well documented problem, resulting in the modelling seen here being informed by the case study "Multirotor Aerial Vehicles: Modelling, Estimation, and Control of Quadrotor" (Mahony, Kumar, and Corke 2012) and the paper "Modelling, simulation and implementation of a quadrotor UAV" (Fernando et al. 2013). The completed satellites will however have 8 motors compared to a quadcopter's 4, and move only in 5 degrees of freedom, (DOF) lacking z translation, (altitude in a quadcopter) as they are confined to the table top.

6.3.1 Inertia

The inertia of the satellite relative to the x axis, J_{xx} , y axis, J_{yy} and z axis, J_{zz} was estimated using a CAD model. These values are then combined into the mass moment of inertia matrix, J_b, equation 11.

$$J_b = \begin{pmatrix} J_{xx} & 0 & 0 \\ 0 & J_{yy} & 0 \\ 0 & 0 & J_{zz} \end{pmatrix} \tag{11}$$

In 3 DOF the only angular rotation is about the z axis; therefore the only relevant value is J_{zz}. The full mass moment of inertia matrix is included here as it is in a standard form, and is utilised in the 5 DOF model (Meriam and Kraige 2012).

6.3.2 Friction

As the contact between the base and the table consists of ball bearings, which exhibit a near static friction behaviour, friction will be modelled as a force proportional to the normal reaction force of the table on the satellite, resisting motion (Dahl 1968). Due to the addition of motors providing thrust in the upwards in the z direction, the normal reaction force, F_n will not be constant, as the thrust forces will oppose the mass of the satellite, m (Meriam and Kraige 2012).

$$F_n = mg - \sum \text{thrust in z direction} \tag{12}$$

$$F_n = m - m\ddot{z}^{ii} \tag{13}$$

The dynamic friction force, F_D is a product of normal reaction force of the table on the satellite,F_n, and the total friction coefficient of the interaction between the glass table and the multiple ball bearings of the base, $_B$.

$$F_D = F_n * _B \tag{14}$$

Due to the circular configuration of the ball bearings,x and y translation and rotation about the axle are opposed equally. When operating in 5DOF a ball and socket joint connects the body of the satellite to the axle. This has been modelled again using static friction, which will oppose angular rotation in all axes equally at the joint. The dynamic friction force at the joint F_J is a product of normal reaction force of the satellite on the joint,F_nJ, and the total friction coefficient of pin and socket joint, μ_J.

$$F_{nJ} = (m - m_{base})g - \sum \text{thrust in z direction}$$

$$F_J = F_{nJ} * \mu_J \tag{15}$$

6.3.3 Free Body Diagram

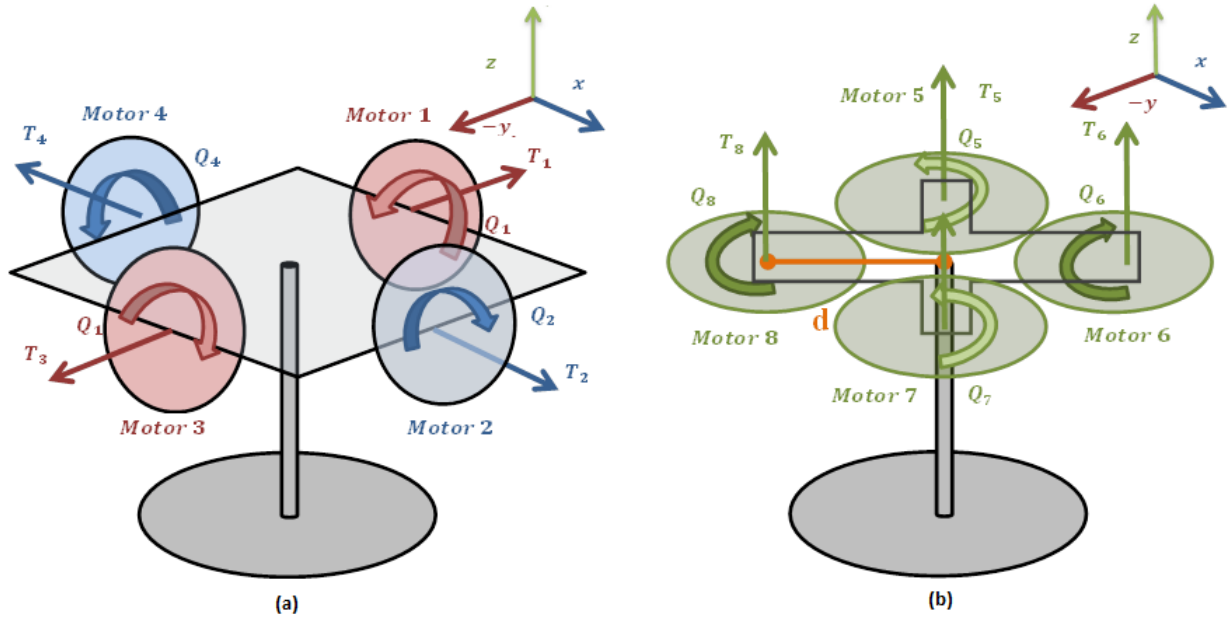

Figure 9: Diagram showing (a) the configuration of the 4 motors orientated to provide thrusts in the x and y direction and (b), the configuration of the 4 motors orientated to provide thrusts in the z direction.

6.4 Equations of Motion

The translation, equation 15 and rotation, equations 16 and 17 of the satellite is determined by the sum of the forces acting upon in the **body frame**, which is centred on the satellite's centre of mass. Variables are assumed to be in this frame: where there is any ambiguity the superscript b is added. The plane of interest is denoted by the subscript x,y or z. Both the 3 DOF equations, equation 16, and the 5 DOF equations, equation 17, are given.

$$F = ma$$

$$acceleration, a = a^b_{x,y,z} = \frac{1}{m}\sum forces^b_{x,y,z}$$

$$\begin{bmatrix} \ddot{x}^b \\ \ddot{y}^b \\ 0 \end{bmatrix} = \frac{1}{m}\begin{bmatrix} \sum forces^b_x \\ \sum forces^b_y \\ \sum forces^b_z \end{bmatrix} = \frac{1}{m}\begin{bmatrix} \sum T_2 - T_4 - F_D \\ \sum T_1 - T_3 - F_D \\ \sum T_1 + T_2 + T_3 + T_4 - F_D \end{bmatrix} \tag{16}$$

$$\text{rotation in x axis, roll} = \phi^b$$

$$\text{rotation in y axis, pitch} = \theta^b$$

rotation in z axis, yaw = ψ^b

$$\text{angular acceleration}, \alpha^b_{x,y,z} = J_b^{-1} \sum torques^b_{x,y,z}$$

$$\begin{bmatrix} 0 \\ 0 \\ \ddot{\psi}^b \end{bmatrix} = \frac{1}{m} \begin{bmatrix} \sum torques^b_x \\ \sum torques^b_y \\ \sum torques^b_z \end{bmatrix} = \frac{1}{m} \begin{bmatrix} \sum 0 \\ \sum 0 \\ \sum Q_5 + Q_7 - (Q_6 + Q_8) - F_D \end{bmatrix} \tag{17}$$

$$\begin{bmatrix} \ddot{\phi}^b \\ \ddot{\theta}^b \\ \ddot{\psi}^b \end{bmatrix} = \frac{1}{m} \begin{bmatrix} \sum torques^b_x \\ \sum torques^b_y \\ \sum torques^b_z \end{bmatrix} = \frac{1}{m} \begin{bmatrix} \sum Q_2 - Q_4 + d(T_7 + T_8 - T_5 - T_6) - F_J \\ \sum Q_1 - Q_3 + d(T_5 + T_8 - T_6 - T_7) - F_J \\ \sum Q_5 + Q_7 - (Q_6 + Q_8) - F_J \end{bmatrix} \tag{18}$$

6.4.1 Relating body frame to inertial frame

The translation of the satellite is described relative to the "ground", or table in the **inertial frame**. Variables in the inertial frame are denoted by the superscript i. The body frame is related to the inertial frame, by using transformation by rotation matrices,R_x,R_y,R_z to take into account the angular rotation of the satellite body, equation 18. (Jirinec 2011)
The rotation matrix which accounts for rotation in all 3 axes is D, where $D = R_x * R_y * R_z$. Due to the usage of sine and cosine functions rotation matrices cannot be implemented if the angles of roll, pitch or yaw will approach +/- 90 degrees. However the limited range of possible motion of the hardware ensures that they are appropriate for use here and this is accounted for in the model.

$$\begin{bmatrix} \ddot{x}^i \\ \ddot{y}^i \\ \ddot{z}^i \end{bmatrix} = D * \begin{bmatrix} \ddot{x}^b \\ \ddot{y}^b \\ \ddot{z}^b \end{bmatrix} \tag{19}$$

In 3 DOF rotation is only possible about the z axis, so the only rotation matrix applied is R_z. This leads to the 3 DOF equations of motion:

$$\begin{aligned} (\ddot{x}^i) &= \frac{1}{m} \cos(\psi)(T_2 - T_4 - F_D) + \sin(\psi)(T_1 - T_3 - F_D) \\ (\ddot{y}^i) &= \frac{1}{m} - \sin(\psi)(T_2 - T_4 - F_D) + \cos(\psi)(T_1 - T_3 - F_D) \\ (\ddot{\psi}^i) &= \frac{1}{J_{zz}}(Q_5 + Q_7 - (Q_6 + Q_8) - F_D) \end{aligned} \tag{20}$$

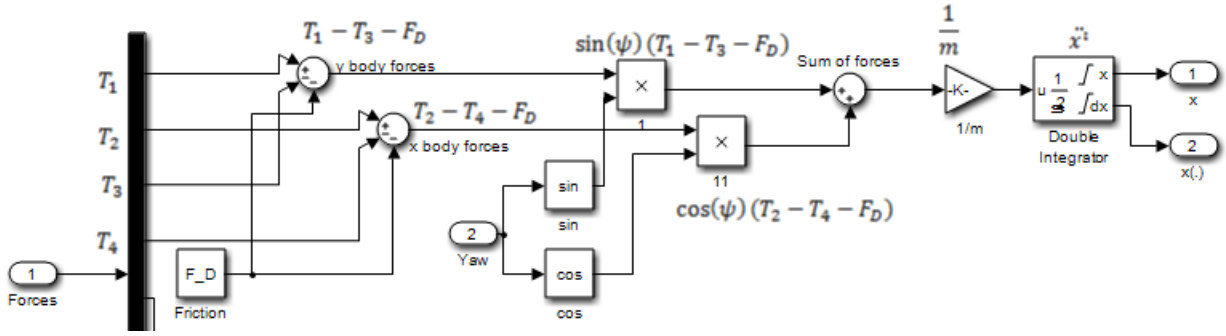

Figure 10: Diagram showing Simulink implementation of the x translation equations of motion.

7 Simulink Modelling Realisation

Simulink was chosen as the modelling software because it is highly modular, allowing for the easy updating of the model for alternative motor configurations, control strategies and different DOF implementations. The current hardware design uses non-reversible motors; however it is possible that future designs may make use of different motor actuation methodologies, providing the ability to efficiently reverse motor direction: this would potentially offer enhanced motion. The model therefore separates the motor control strategy from the control algorithm, allowing for easy updating for future use and facilitating the comparison of different hardware configurations, ensuring optimal implementation.
Modelling of specific system behaviours is discussed in the body of the report; the structure of the Simulink model itself is shown in Appendix J.

7.1 Control

7.1.1 Hardware Control Strategy

In figure 11 we see part of the Simulink implementation of the hardware configuration: in the current configuration, thrust in the positive x direction is provided by motor 2, and in the negative x direction by motor 1. The code above sends the x control signal to motor 2 if the x control signal is positive and to motor 4 if it is negative. This separation of the hardware control strategy from the control algorithm allows the easy modelling of alternative hardware configurations, and easily accounts for the use of both reversible and non-reversible motors.

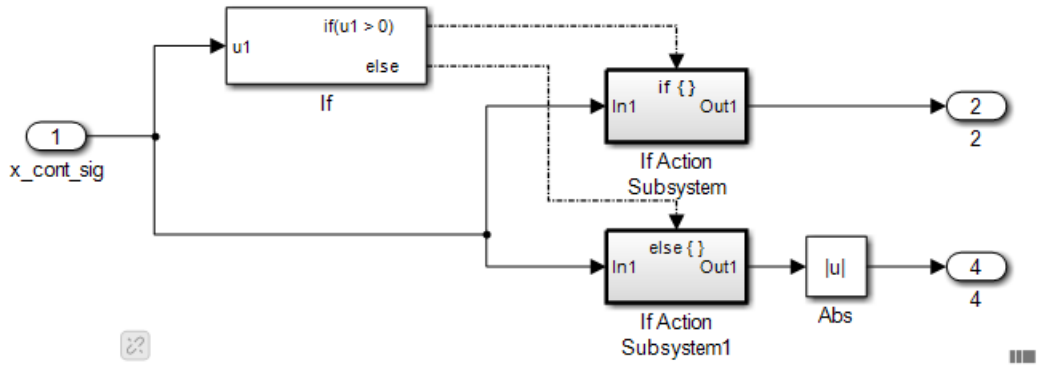

Figure 11: Diagram showing Simulink implementation of the current hardware configuration for movement in the x axis.

7.1.2 Control Algorithm

Reference signals in form of desired x, y and yaw path will eventually be inputted from the swarm position algorithm; however in the simulation they are a predefined set of step coordinates designed to be similar to a probable path, and to excite both positive and negative x and y motion.

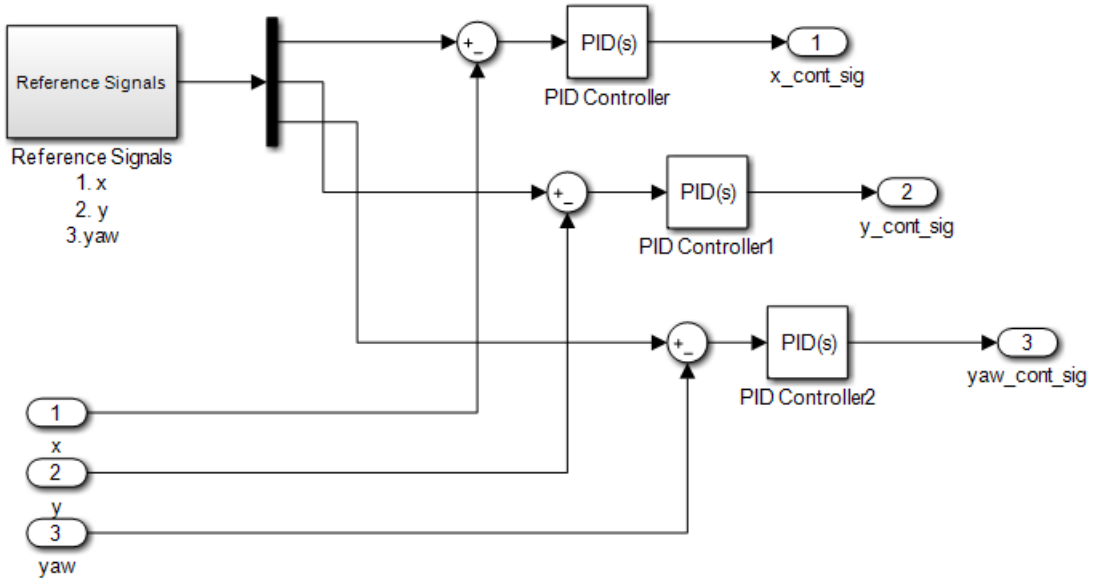

Figure 12: Diagram showing Simulink implementation of the current PID control strategy.

For the 3 DOF model a simple Proportional Integral Differential, (PID) controller for each degree of freedom, as seen in figure 12, is sufficient. However when implemented on the 5 DOF model this control strategy fails: this failure is due to the 5 DOF system being highly coupled, as motors are used to actuate rotations in multiple axes. This coupling results in the need for a more robust control strategy. The controllability of the satellites could also

be improved by designing an updated 5 DOF hardware configuration. (Argentim et al. 2013).

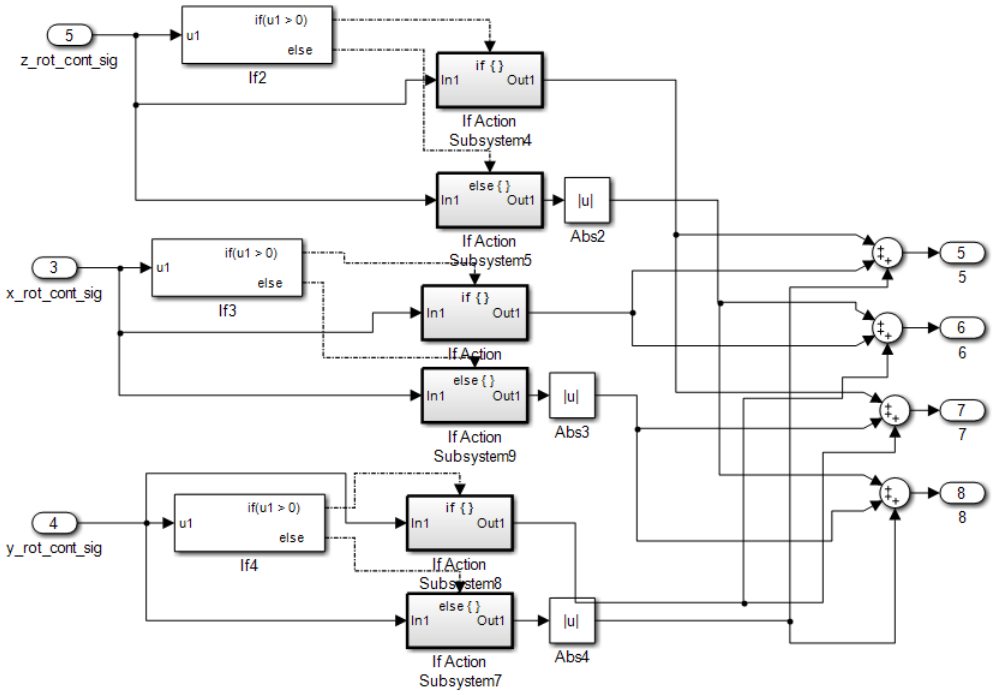

Figure 13: Diagram showing Simulink implementation of the current hardware control strategy implemented in 5 DOF.

Tuning of the control PID parameters was carried out both by utilising the MATLAB automatic tuning tool and using knowledge about the system's behaviour. The responses are compared below: which controller is implemented depends ultimately on the swarm algorithm proposed, and whether offset or overshoot are more detrimental to safety. However as the response time requirement of the system is flexible due to the satellites not needing to move quickly, it will certainly be possible to obtain a response without offset and overshoot on the real system.

7.2 Simulation Results

The satellite response was simulated with an input reference signal typical to path which will be required when operating. A response which almost exactly followed the reference was introduced. Due to the system having quite generous step time requirements it proves possible to achieve an offset and overshoot free response. Additional random disturbances were added to the forces on the satellite to simulate the disturbances introduced by having multiple satellites on the same table, allowing for the possibility of thrusters blowing on nearby satellites. Additionally to determine the performance of the yaw PID reference

tracking, which in the first simulation was not sufficiently tested. Additional overshoot is introduced, but the overall response is still acceptable.

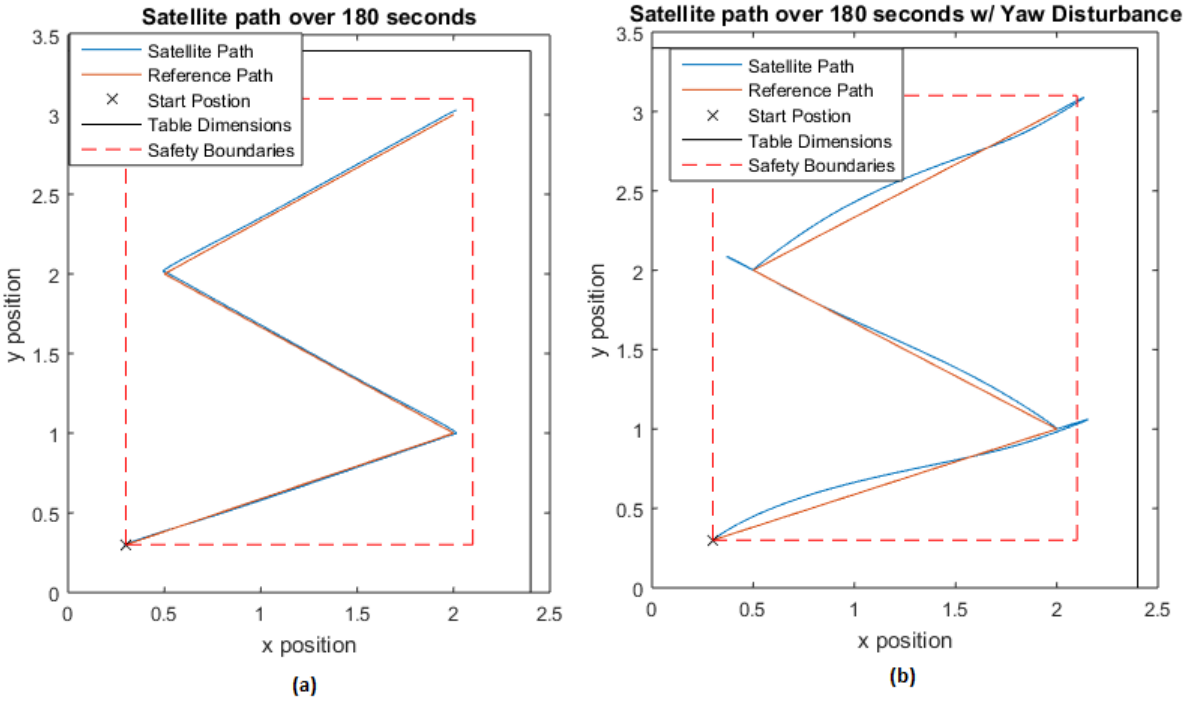

Figure 14: Diagram showing the path of the satellite, (a) corresponding to the response in figure 15, and (b) corresponding to the response in figure 16, (with additional random disturbance.

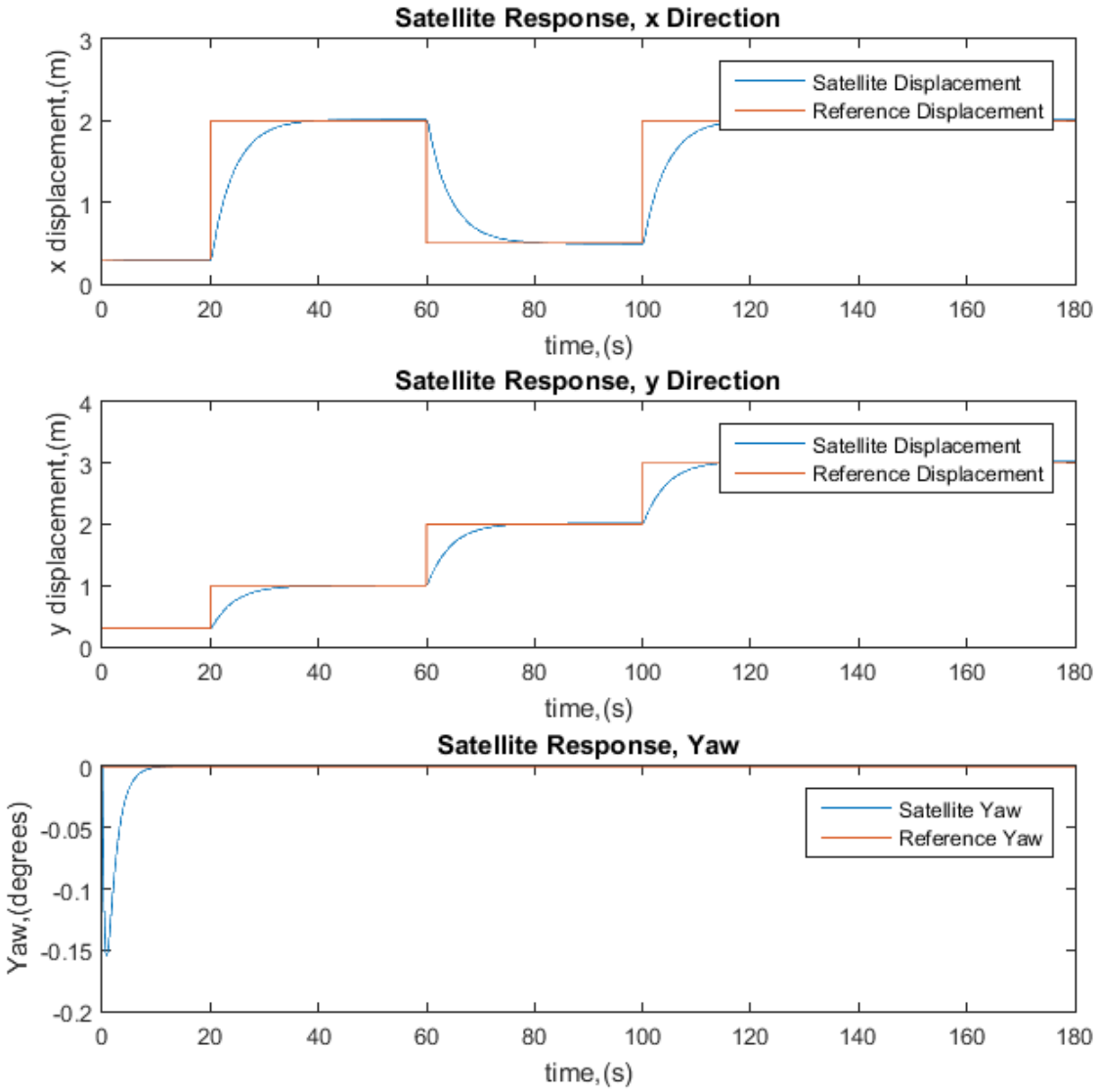

Figure 15: Graphs showing the x, y and yaw responses of the satellite model to a x and y position reference and a constant yaw=0 reference.

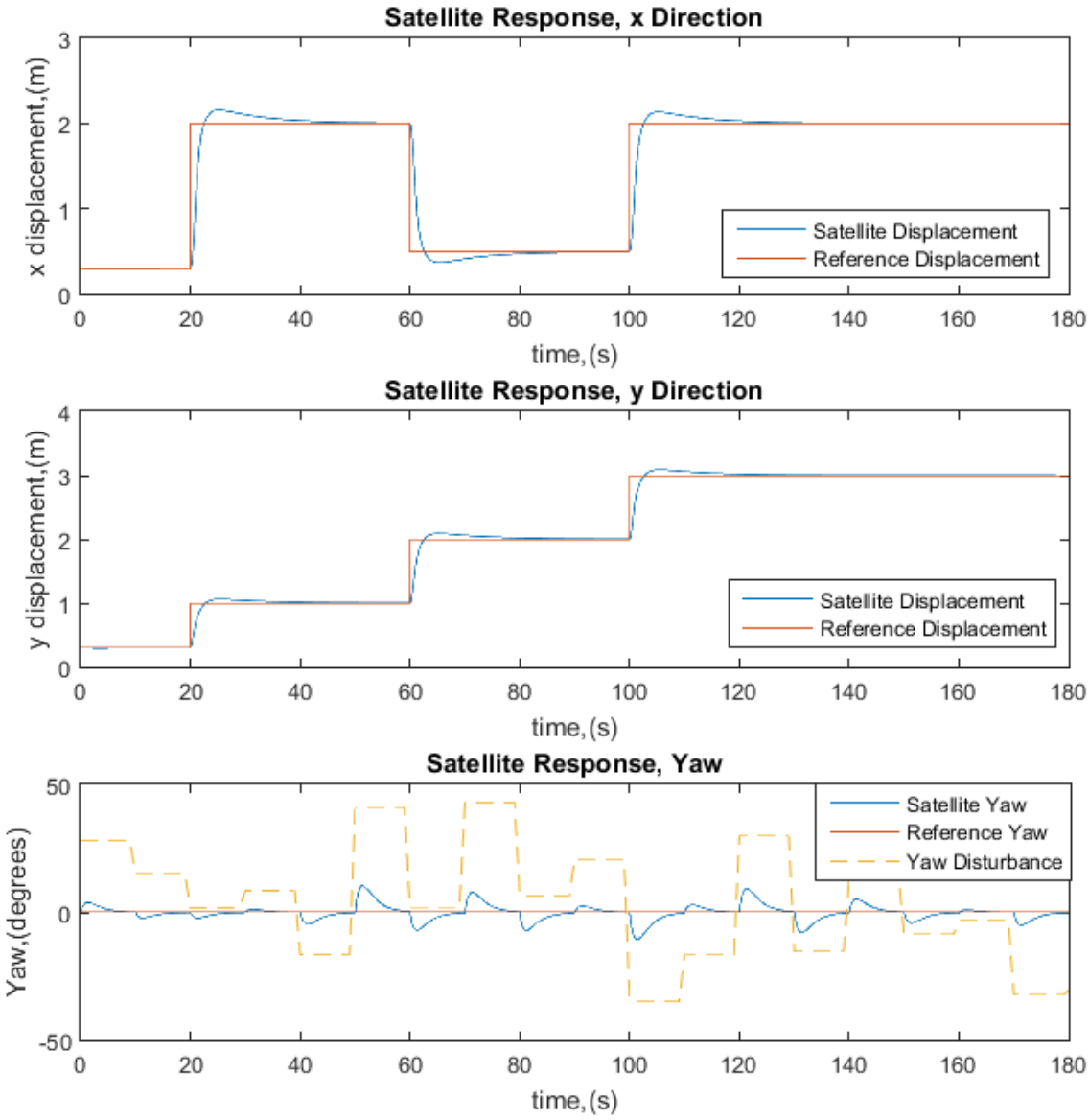

Figure 16: Graphs showing the x, y and yaw responses of the satellite model to a x and y position reference and a constant yaw=0 reference with additional disturbance.

8 Computer Vision

8.1 Why use Computer Vision?

With clusters of satellites, it might be easier to instruct them to move according to a fixed object in their surroundings (such as a landmark or a fixed marker) as opposed to having them move relative to one another (the main issue with this is always adjusting motor velocities and travel angles for every single satellite differently, and not having an absolute position at any time).

This can be solved by implementing a computer vision system. Computer vision allows us to transform or process data from images or from a video feed (usually from a camera), and design algorithms to extract specific information from the images (Bradski and Kaehler 2008). Therefore it is possible to control the group of satellites by attaching a set of cameras to each of them. Coloured or patterned markers, black and white shapes and even QR codes could be used through recognition algorithms to allow the satellites to map an area or locate where they are in the space around them. By using computer vision, it is possible to obtain distance and position estimation all from one or a set of cameras, as opposed to having to install several different sensors on the same system.

By fusing the visual feed from the cameras with the sensing data from the gyroscopes in the system, it might even be possible to perform attitude control when the satellites are modified to move in five degrees of freedom (Lincoln and Veres 2006).

8.2 ROS & OpenCV

Matlab has functions which could be used to do the computer vision where it will do multiple different commands to get the output that is desired however you cannot edit these functions to make them work in the way which is desired. The Odroid also only works using Ubuntu and has too little computational power to run Matlab. If it were capable of running Matlab then many of the additional toolboxes would be needed in order for the code written to be transferred across to the Odroid in a way that the Odroid could execute the commands.

ROS stands for Robot Operating System and contains many open-source libraries which are useful when coding robotic systems. ROS was designed so that Object Orientated Programming techniques could be used in developing functions that are modular, which was desirable for this project (Garage 2016). ROS employs a distributed systems method, where it uses nodes (processes that carry out required computation) to control individual features in the system. There are several benefits to using ROS nodes, such as increased fault tolerance (as faults can be isolated to individual nodes), and reduced complexity (Foundation 2016b). ROS nodes can be combined to stream ROS topics, whereby a publisher/subscriber system can be set up, similar to a client-server relationship. This publisher/subscriber system is used as the basis of the computer vision software in this project, mainly to set up a link between the master PC and the camera module, as seen in Appendix E.

In order to use OpenCV on ROS, it is necessary to set up a working environment called the catkin workspace (which essentially is a standalone workspace for developing software), which allows the user to define specialised packages (known as metapackages) that can be used with other packages (Foundation 2016a). This is very useful for this sort of project, as it allows us to create a workspace in which the software can be developed and built, and within which all libraries and software dependencies can be seamlessly linked. This is done by modifying the Cmakelist.txt & package.xml files generated in the workspace to make sure all the OpenCV dependencies are extracted from the appropriate ROS packages, which are in appendices x and y.

8.3 Image Processing Methods

A few commonly used computer vision and image processing methods are briefly discussed below, following which a decision about which method(s) to implement is made.

8.3.1 Quick Response (QR) Codes

QR codes were considered as they are rotationally unique so it would not matter what the orientation of the satellite was it would still read an exact position. QR codes also maintain appearance when seen at an angle which means they can still be identified. This is not the case with an image such as a square where if you look at it from an angle it can appear to be a trapezium. This is done by having identification markers in three of the corners so that if the image is rotated then the correct orientation can be identified and also when viewed at an angle the relative sizes of the areas of each square is the same (Harath 2016)(Sinha 2016).

The main issue is that within ROS and OpenCv there are no native libraries for QR codes. This would mean that functions would need to be created from scratch on how to read the QR codes which would have created a lot more work to do that is complex in nature. Also the majority of information about developing QR code readers is for use on phones using some of the software already installed which would not be available when writing the code.

8.3.2 Coloured Patterns

OpenCV has an extensive amount of inbuilt functions designed to detect, modify and handle colours in images and video frames. Methods such as HSV and RGB are very commonly used and are therefore a safe choice for this project as documentation is readily available. For instance, patterns, each of a different colour or combination of colours, could be used as landmarks and detected by the satellites.

8.4 Chosen Methods

After consulting the project advisor, Dr Jonathan Aitken, it was decided that the best solution was to design a colour and edge detection system that could be used with a grid suspended above the glass table, on which specific shapes of varying colours are printed, each

of which represents a different location (The final grid design can be found in section x). The camera used is the e-CAM51,an autofocus camera module with a 60° viewing angle & which compatible with the Linux OS being used, in this case, *Ubuntu 14.04*, on which ROS & OpenCV run. The camera sits on the top of the satellite, facing the grid that is suspended.

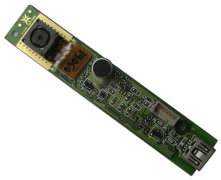

Figure 17: E-Con Systems 5 MPixel USB 2.0 Camera (Systems n.d.)

8.5 Colour Segmentation & Detection

As discussed above, the initial stage of the computer vision process involves detecting specific colours. In order to do that, two options were available:

- Red-Green-Blue channel (RGB) processing
- Hue-Saturation-Value (HSV) processing

The choice was fairly easy to make. While using RGB channels allows for easy detection of the three primary colours, using to HSV allows for a much finer control over colour detection. With the use of the HSV channels, it is possible to exert control not only on Hue (which allows for the detection of a wider range of colours than RGB), but also Saturation and Value (which can also be referred to luminosity). This is of particular importance here, as lighting conditions in the laboratory can sometimes act as a potential source of noise, or distortion and it would be desirable to have a certain amount of control on these parameters too. In addition to this, HSV processing, when applied in the context of detection, can be less taxing on memory space and computational complexity as compared to RGB processing (Qu et al. 2013).

OpenCV captures RGB frames from the video stream by default. These are converted to a HSV image using the inbuilt command `cvtColor()` (with specified parameters), which is a conversion function. In order to control the HSV channels so that a specific colour can be detected, a control window containing a trackbar for each parameter is created. The trackbar allows this colour segmentation process to be even more intuitive, since the output is simultaneously updated on screen. As the trackbars are tuned, they create a threshold of acceptable values for each channel. These thresholded values are used to compute the thresholded image using the `inRange()` function. While these are not filters per se, the trackbars provide a threshold level, acting in a similar fashion to a bandpass filter. Note that the camera is set to operate at a 20 Hz framerate (20 FPS) in order to obtain a smooth response on screen, even when the camera is being moved around.

The figure 18, below, shows a test code used to detect a red triangle through the HSV method. The code can be found in Appendix F.

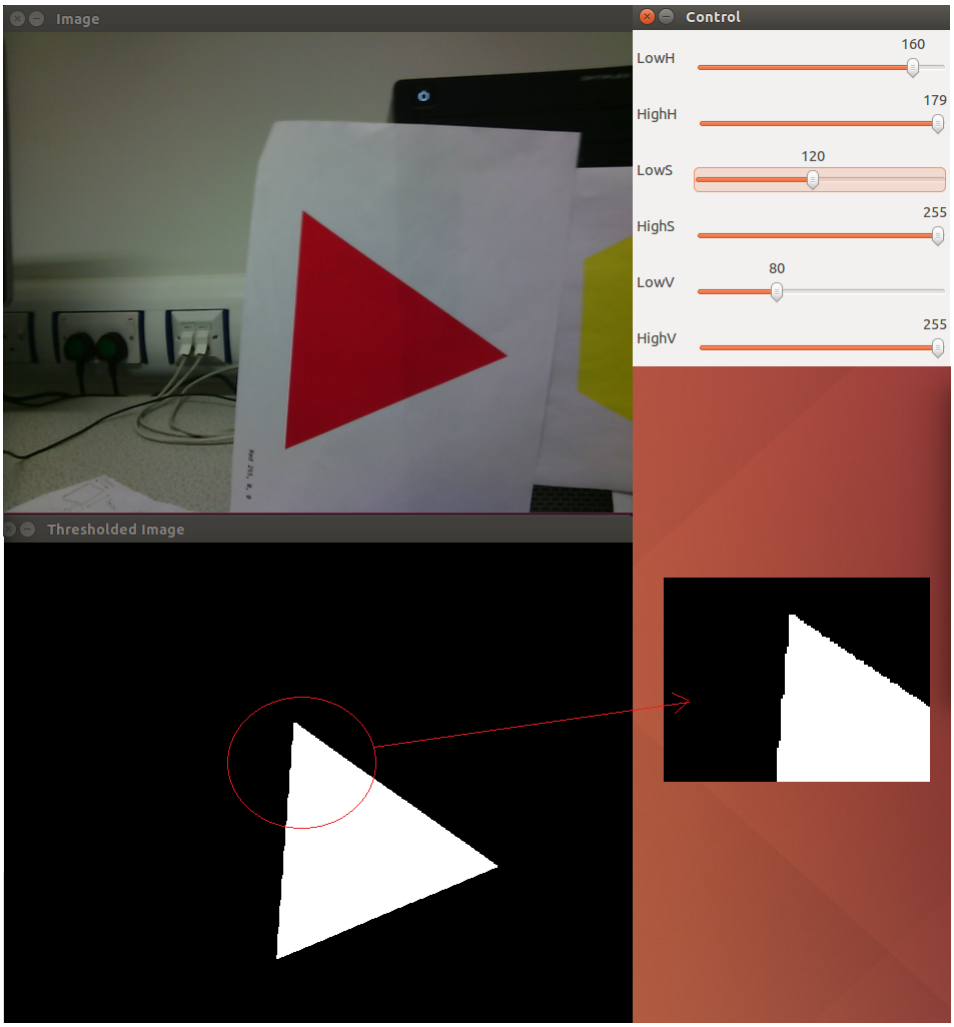

Figure 18: (a) RGB Frame on top, (b), Tuned HSV Control Window, (c) HSV Thresholded Frame on bottom, with zoom on the edge

As expected, by tuning the HSV values (Hue is set to 160-179 which is the range for the colour red, and Saturation & Value are tuned manually to match the room's lighting). However, the zoom on the thresholded image reveals that the image is still quite grainy, which is an indication of high frequency noise. As such, noise filtering methods are discussed in the next section.

8.6 Smoothing Filter Implementation

At this stage, noise is still very much present in the thresholded HSV output, although that may not be directly apparent to the human eye. Before moving on to edge detection (also known as contour detection), it is therefore of utmost importance to filter out the noise in order to avoid any ambiguous output.

As such, a smoothing filter will be implemented. A very common filter for this sort of operation is the Gaussian Blur, which is a low-pass filter. A Gaussian blur filter operates by applying a Gaussian probability density function to blur, or smooth a grainy image by removing high frequency noise. Effectively, it simply removes details and softens an image. In mathematical terms, it works through kernel convolution - it convolutes a small kernel of size *nxn* defined by the Gaussian function to the image matrix, whose size is usually much bigger than the kernel matrix (Gedraite and Hadad 2011). Here, a kernel of size *5x5* has been defined. It is relatively small in size as we wish to conserve the edges of the shapes in our image. It is also preferred to use a small-sized kernel to reduce an otherwise lengthy computational time (Waltz and Miller 1998).

While a Gaussian blur does smoothen an image and filter out a lot of the high frequency noise, it is usually necessary to apply some basic morphological operations to get rid of small objects that could be interpreted as noise later on. In the context of image processing, a morphological operation is simply a quantitative analysis of the geometrical structure of an image. Non-linear filters designed through this ideology usually have a high algorithmic efficiency (Maragos and Pessoa 1999). In this case, we simply apply the process of eroding/dilating the image (followed by the reverse, i.e. dilating/eroding). This step ensures that small objects or irregular intensity changes that are scattered through the image frame are isolated, removed and then filled in in order to further smoothen the image (OpenCV 2016).

The figure 19, below, shows the results of applying a Gaussian Blur & Morphological operations to the HSV output. The code used to obtain this output can be found in Appendix G.

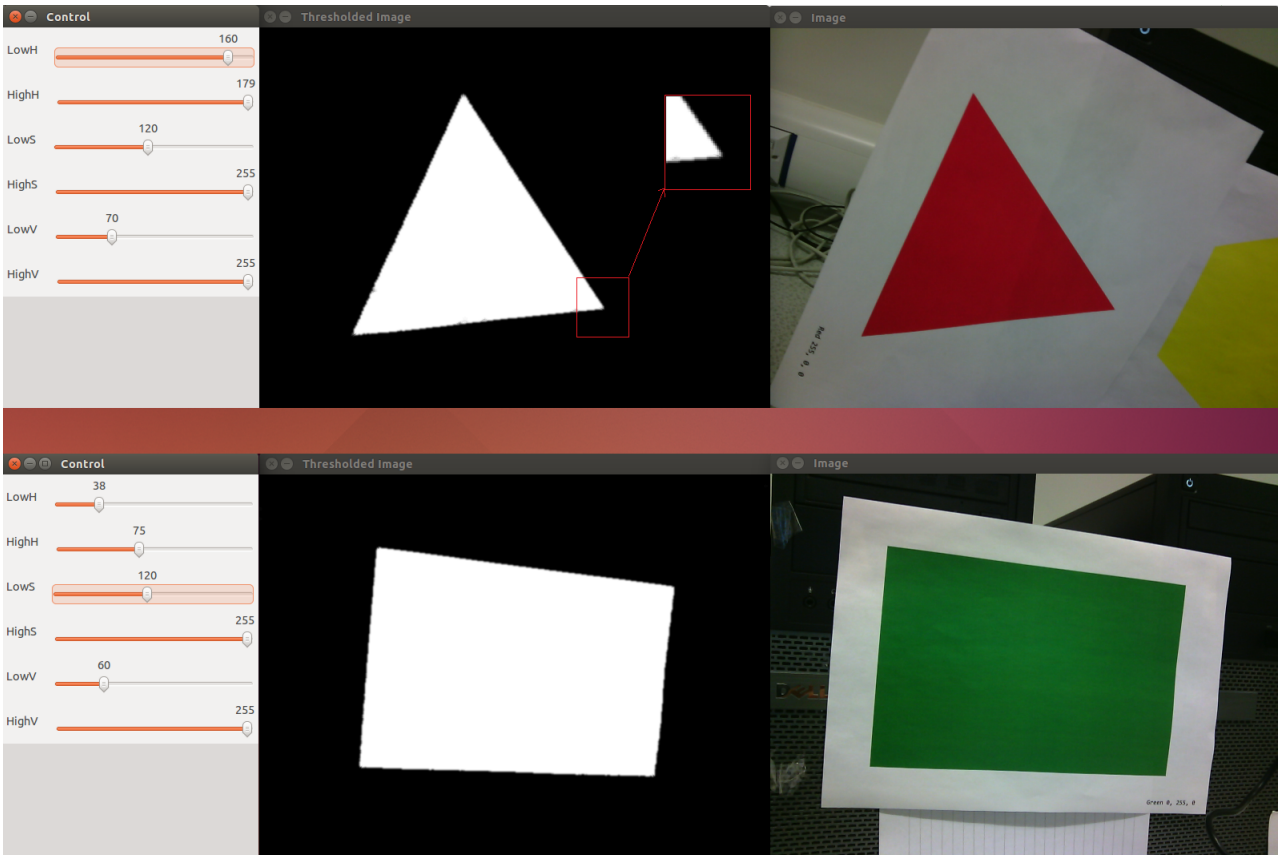

Figure 19: (a) Dataset showing smoothing filters applied to red threshold, with zoomed section (top), (b) Dataset showing smoothing filters applied to green threshold (bottom)

It is obvious that applying a Gaussian blur and an erode/dilate operation result in a smoother output, without affecting the quality of the edges, which will be detected in the next section.

8.7 Contour Detection

The next step in the computer vision section is to detect the edges (contours) of the coloured shapes filtered through the HSV method described above. Before that, some more advanced filtering techniques designed for edge detection are discussed below, and their viability in the scope of this project are also assessed.

8.7.1 Advanced Filtering

Contours can simply defined as curves joining continuous points that have the same intensity or colour on an image (OpenCV3 2016). They are effectively what the human eye perceives as edges of objects. In computer vision, there are sets of non-linear filters specially designed for edge or contour detection. The Sobel and Canny edge detection filters are viable options

that can be used to detect the edges of shapes. It is necessary to convert the images into a binary format before applying such filters - These filters attempt to connect points with the same colour or intensity, which means using a binary (black and white only) input increases the efficiency of the process.

The Sobel filter is a first order filter that works in a similar way to the Gaussian blur described earlier (i.e., it also applies matrix operations similar to the kernel convolution). The difference is that Sobel works in the x & y directions in order to preserve as many edges as possible, but that also means that it tends to be fairly noisy as it picks up several noisy segments that are wrongly interpreted as edges (Beeran Kutty et al. 2014). As it works in the x & y direction, Sobel tends to give a certain thickness to some edges that are detected in both directions.In order to solve this issue, the Canny filter can be used. One common version of the Canny filter uses the Sobel output as its input. The Canny algorithm applies two levels of thresholding. It first applies a kernel operation which reduces the thickness of all detected edges to a width of one pixel. The second level, also known as hysteresis thresholding, verifies that detected pixels are connected to form a contour, and filters out pixels with low magnitudes which, effectively, is just noise (Ogawa, Ito, and Nakano 2010).

A matlab script (found in Appendix H) was used to apply the Sobel & Canny filters on a test image as shown below.

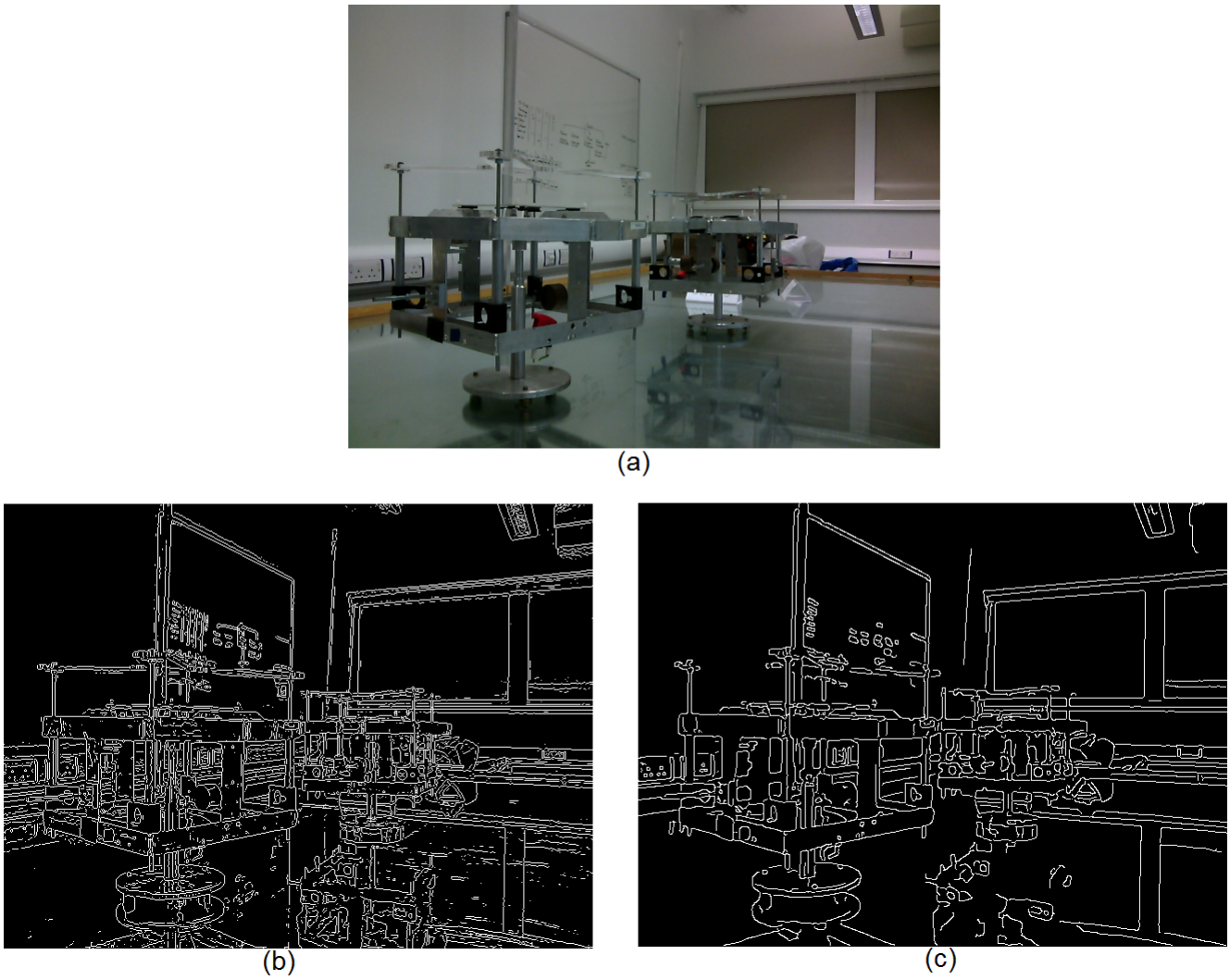

Figure 20: (a) RGB Colour Frame showing the original image, (b), Sobel Filter Output, (c) Canny Filter Output

As expected, the canny filter contains thinner edges and less noise too. However, the use of these filters can be completely circumvented during the contour detection process in the scope of this project. The reasons behind this choice are outlined in the implementation section next.

8.7.2 Contour Detection Implementation

As just mentioned above, the use of edge detection filter such as Sobel and Canny can be complete avoided, and this choice is supported by the following explanation:

- By using the HSV thresholded output (described in processes above) as an input for the contour detection process, the need for complex filters involving more kernel calculations (which can be potentially computationally taxing when running at 20 FPS) is totally

circumvented & a relatively clean result is obtained, provided S & V have been tuned, as shown in figure 22 (a) and (b) below.

- Only polygonal (i.e. geometric) shapes, each of uniform colour, were included in the design for ease of detection and computation - even circles are eliminated from the design, as these would require a separate hough transform process to be detected as they have no connection vertices or edges (Chen and Chung 2001). Only detecting objects of low complexity supports the choice of not using advanced filters.

OpenCV has two inbuilt command called `findContours()` & `drawContours()` which contain algorithms that already handle basic filtering and edge detection, and display them when given a binary output, respectively. figure 21, below, shows the use of these two functions. A normal RBG frame is converted to binary, which is then processed by `findContours()` & `drawContours()`. As no specific HSV values have been filtered here, the code therefore attempts to pick up and display every single contour that it can detect. We can conclude that these functions can be also applied to simpler shapes of uniform colour.

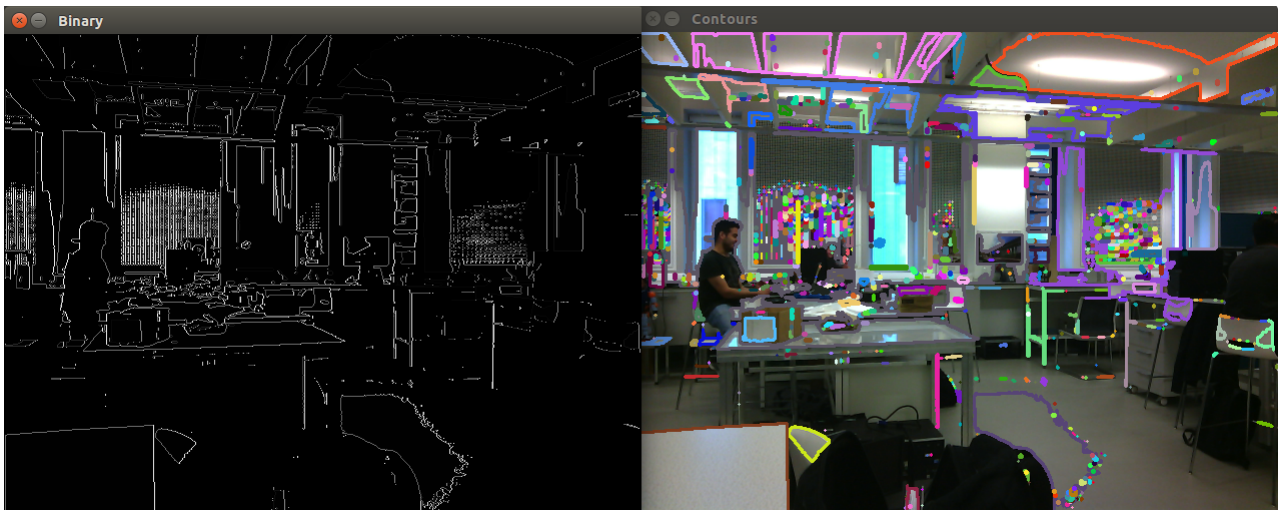

Figure 21: (a) Binary input, (b) Contour Detection output showing every contour on the frame

During the colour segmentation process, a thresholded (or filtered) HSV output is obtained, which is a black and white image only showing the required colour (for instance, see figure 19 in section 8.6 above). This thresholded HSV image is converted to binary format and passed on as an input to the contour detection algorithm used above, which results in contours of the thresholded HSV polygons being detected and drawn on a separate frame. Two examples are shown in the figure 22 below, where a blue pentagon and red triangle are detected using this method.

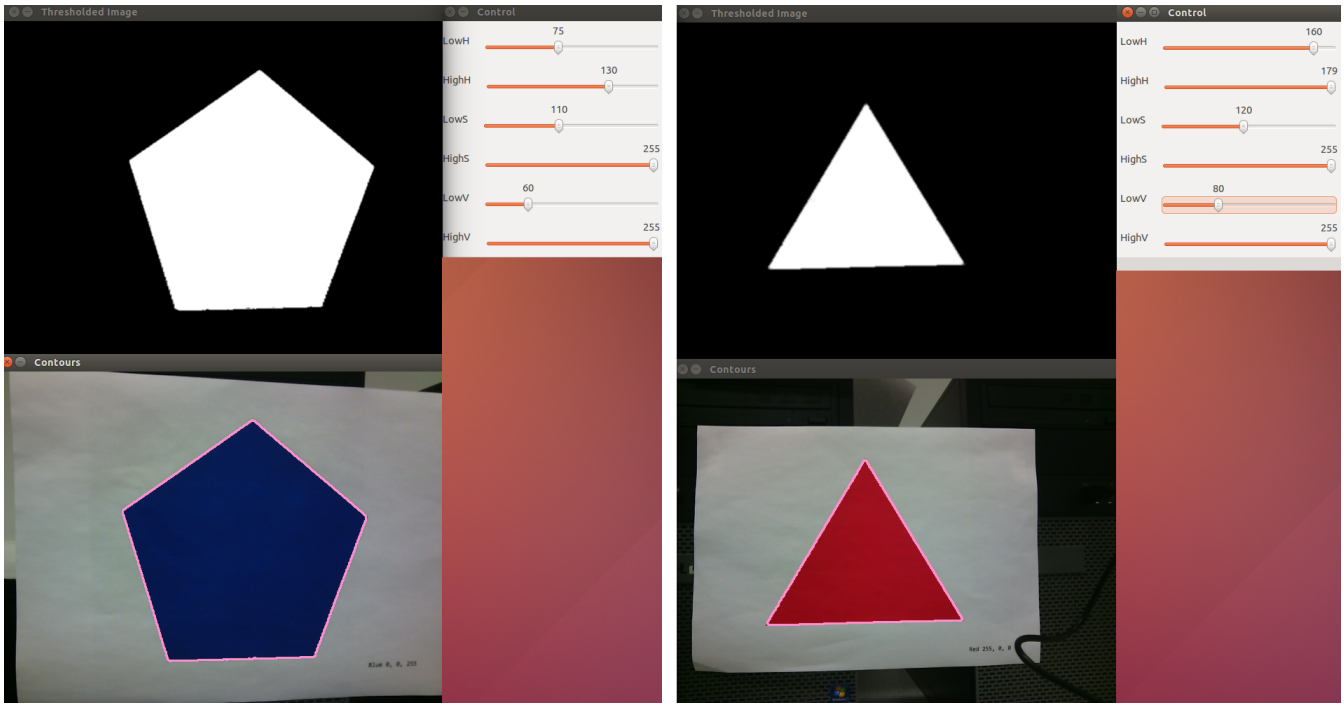

Figure 22: (a) Blue Pentagon detection (left), (b) Red Triangle detection (right)

Similarly to the HSV section above, it is also possible to filter colours and detect contours in real-time even if several shapes of different colours are present on the same frame at once. This is enabled thanks to the trackbars controlling Hue, Saturation & Value, which pass the trackbar values to the detection algorithms in a while loop. The code used to obtain this result can be viewed in more detail in Appendix I. An example of detecting a specific shape/colour while many are present in the same frame is shown in figure 23 below.

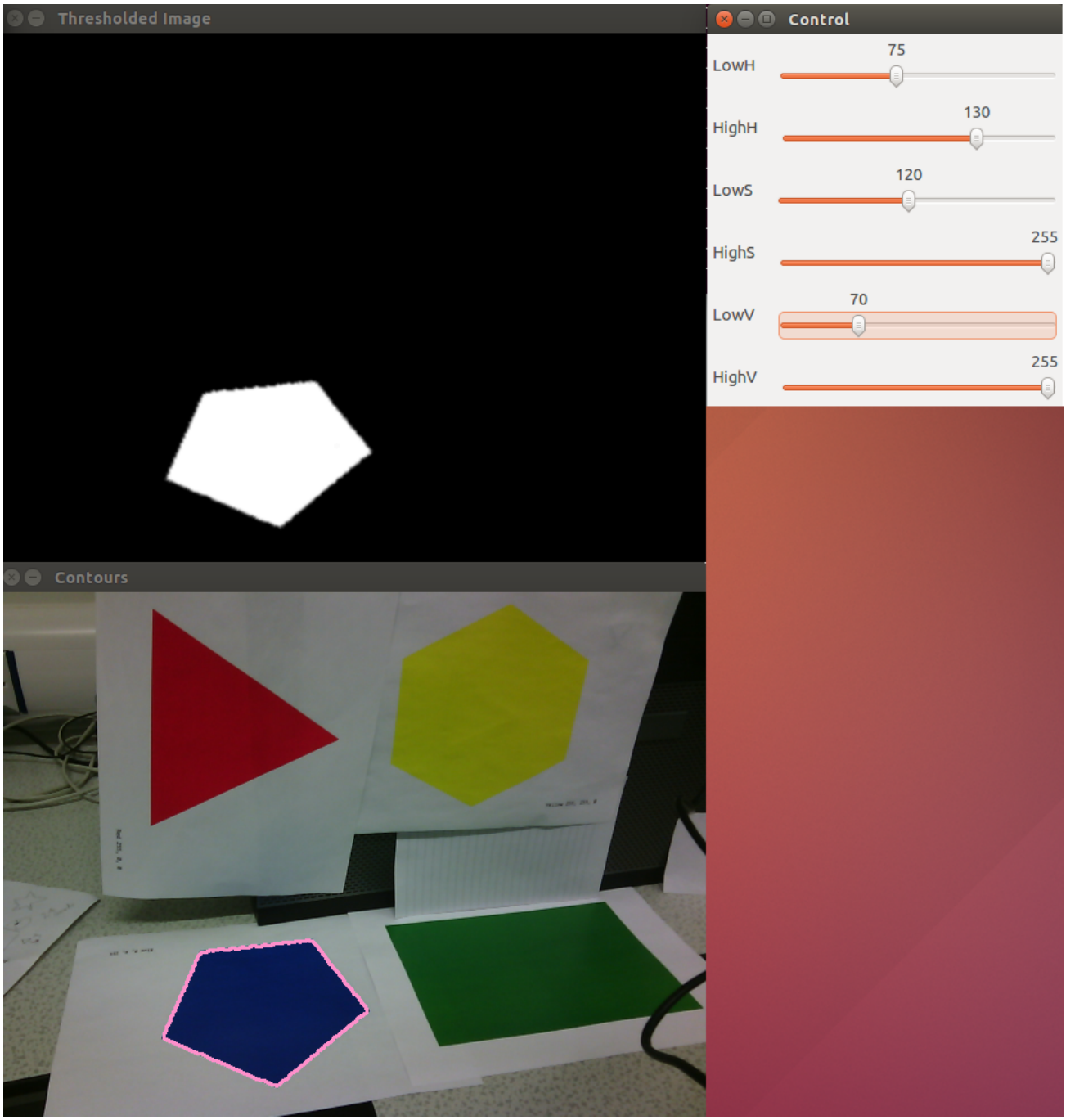

Figure 23: Detection of only one shape of the specified colour

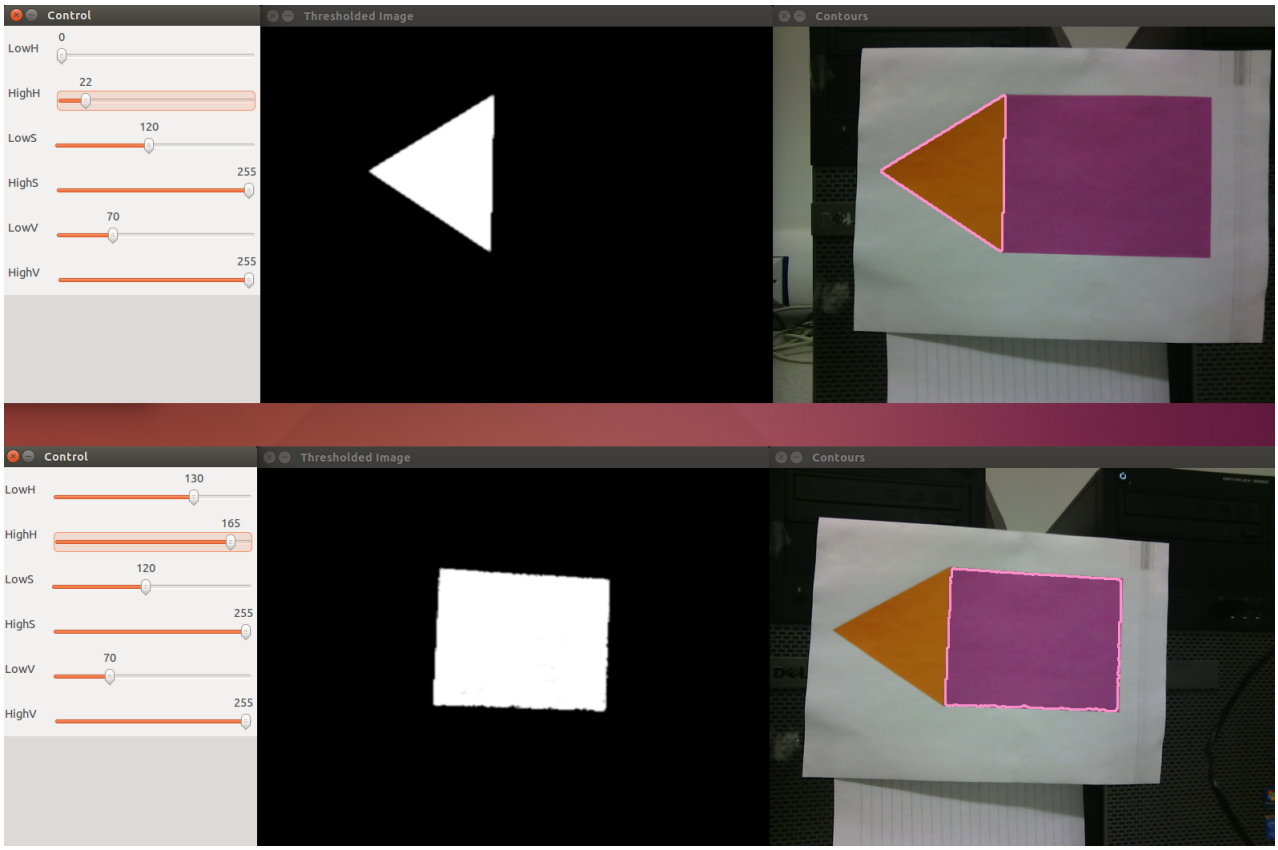

Figure 24: Code Robustness shown by composite colour & contour detection - (a) Orange triangle (above), (b) Violet Quadtrilateral (bottom)

Figure 24, above, shows the robustness of the code - As HSV was used instead of RGB, the program can easily be used to detect composite colours such as orange or violet, and even manages to detect their contours with an acceptable level of accuray. This could potentially be harder to achieve through RGB, as, for instance, violet is composed of a ratio between the Red and Blue channels. The contour detection process is also fairly robust, as it manages to detect both the orange triangle and the violet quadrilateral separately even though they share an edge.

A simplified flowchart of the processes and decisions made by the final computer vision software is shown below.

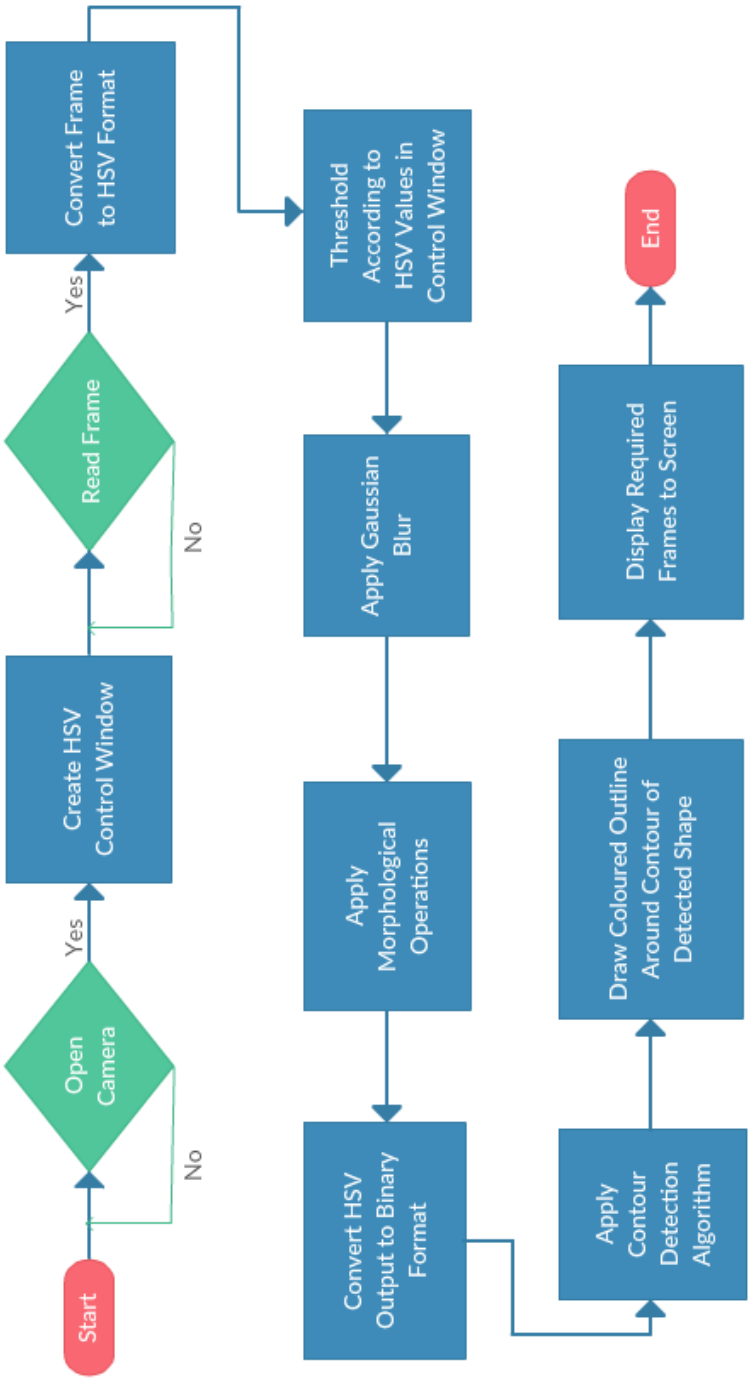

Figure 25: Flowchart of Computer Vision Software Implemented on OpenCV

9 Autonomy & Localisation

9.1 Localisation

In order to be able to control the positioning of the satellites, the actual position of each satellite would be needed, so as to stop collisions between satellites and the edge of the table. This will also allow the satellites to compare their expected position.

9.1.1 Global Positioning System (GPS)

Coming up with the method for location was one of the first challenges and a variety of methods were considered. The first option was to use triangulation in order to obtain position as this is the current method used to locate satellites as it is accurate over long distances. This uses GPS signals and reads the time from the different stations from which this is being transmitted and the time difference is calculated between at least three different stations so the position from that can then be calculated. This position is then compared to where the satellite is meant to be and the satellite aims to move to that position (Sheynblat 2000) (Lau 1995) (Class and Hartman 1996).

Accuracy is relative and for satellites thousands of kilometres from the signal transmitter then the accuracy is to within a few metres of the actual position of the satellite. When scaled down to the table top satellite system being used, this method becomes impractical as the accuracy level would not be sufficient as the table is only a couple of metres in across and therefore this would not provide a safe enough prediction for which the satellite could use to compare its position to the position of one of the other satellites.

9.1.2 Mapping

The second solution to this problem that was considered was to use a grid that would be mounted above the table, the grid would have markings all equally spaced and then the satellite would be able to look up at the grid and be able to tell its position from that. Mapping would mean that the satellites would send their information to the computer which would then build up a map of the area around them and eventually map out the table so they could identify any point that they are at on the table. The disadvantage of this is that it would be difficult to make the algorithm that would create the map.

9.1.3 Grid with Look-Up Table

The second grid design was to have individual markers on the grid and then have a look-up table so that the satellites could look at this and then be able to tell its position from this. Having a look-up table would make the computational requirement a lot easier as it just needs to be databased beforehand. This grid would have to have markers on it that are rotationally unique as the satellites might spin on the table so if there was rotational symmetry between two of the markers then the satellite could identify the incorrect position on the table.

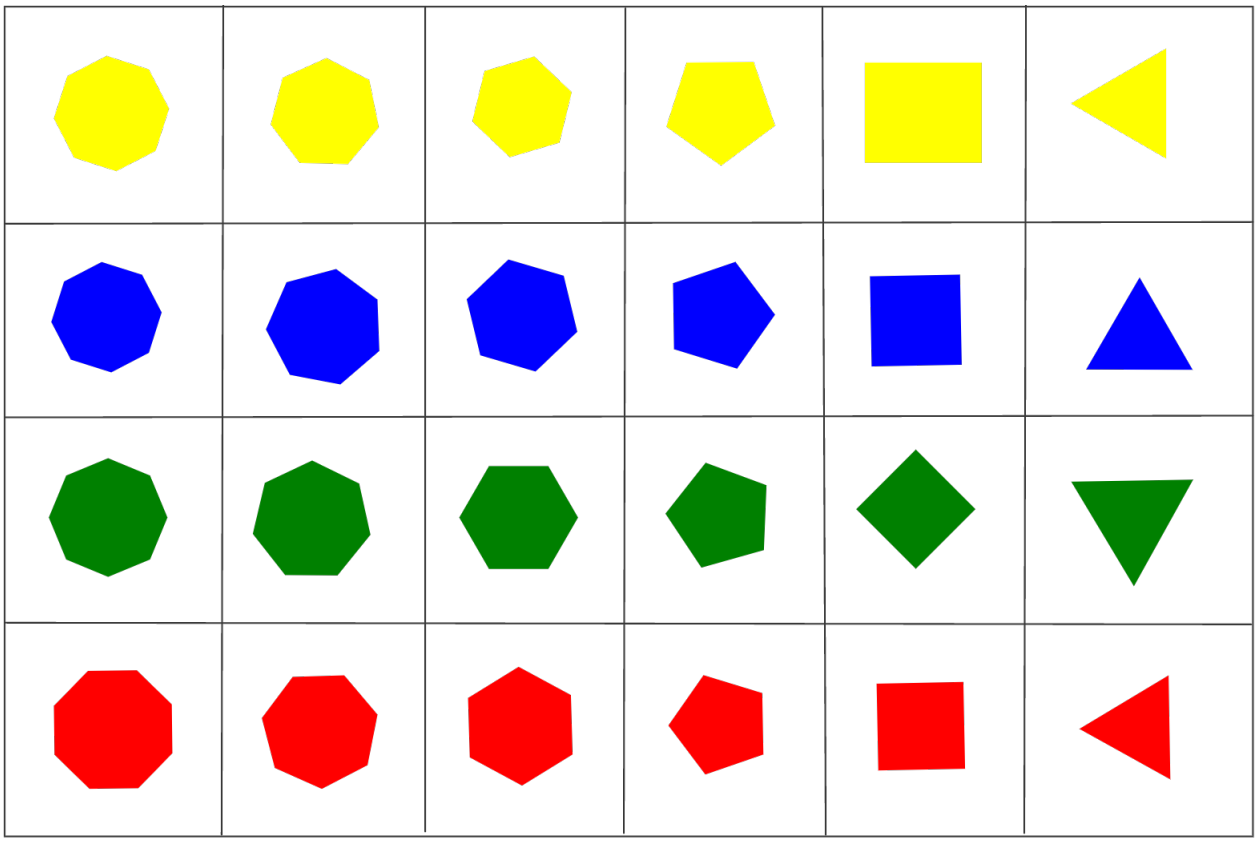

Figure 26: Final Grid Design

The final grid design was chosen to use six different shapes, depicting the x axis, and four different colours, as the y axis. This would allow the satellites to locate themselves easily as if it identifies that the main part of the image is the colour blue and if it saw a pentagon then it would be in position (4,3). Due to the set up of the grid the rotation on the satellites would not matter as it needs to identify the number of contours that it can see in order to get its position rather than identifying a specific shape. By using different for each image there is no rotational symmetry within the grid so there is no possibility for confusion as a certain colour will always depict the row that the satellite is in.

9.2 Autonomous Formation Movement

The satellites are required to move in formation as part of the brief. There are a few different solutions that would allow formation movement. This links to the idea of getting satellites orbiting the Earth to be able to move in formation to perform more complex tasks or to do tasks to a better quality than an individual satellite.

9.2.1 Master-Slave configuration

One of these solutions would be to have a 'master' satellite which knows its position and then instructs the other 'slave' satellites to go to certain positions relative to it in order to

maintain the formation. This would make the master satellite the focal point of the formation (Saska et al. 2014). This was considered and then the idea was rejected as it would be much better for the satellites to act independently so that if one of them were to fail then then the others can still operate and perform the expected task (Merrill and Becker 2015).

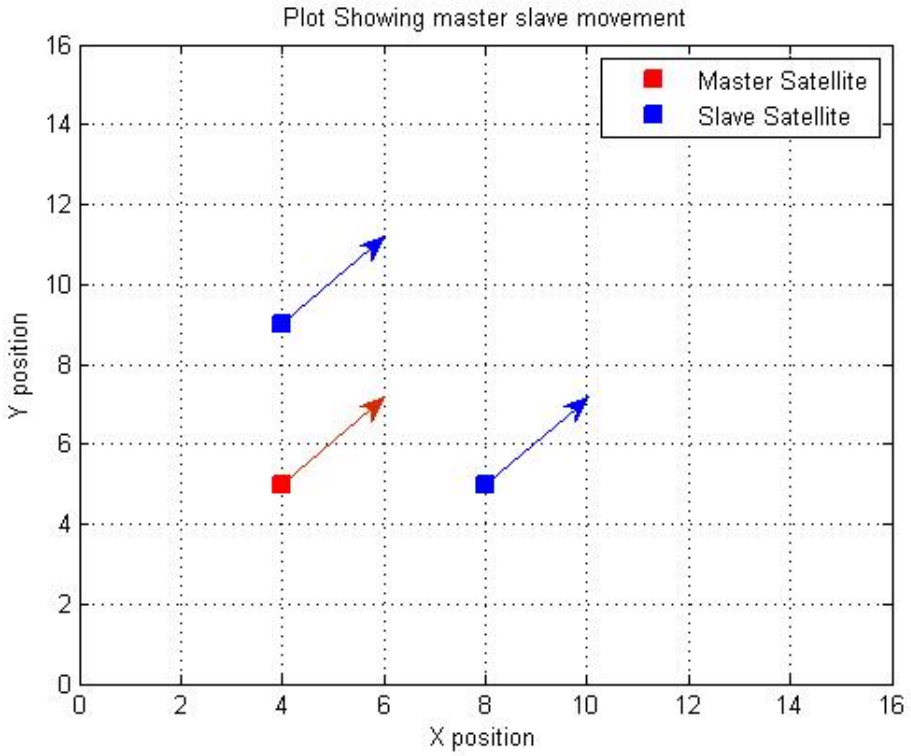

Figure 27: Simulation showing master slave movement

Figure 27 shows how the satellites maintain a set distance from the master satellite and would then move with the master upon its command. The distance the satellites maintain is between the slave satellite and the master and not necessarily between the two slave satellites. If the two slaves were to get too close then it would require the master to understand this and tell them to move apart, otherwise the slaves could have an override program that runs when they get too close to one another to get them to move apart or to shut down (Hesselmar and Brodin 2015).

9.2.2 Centralised Focal Point

Another solution was to have a centralised focal point for the formation and then all the satellites will move to a set distance away from the point and from each other as they will send positional information to one another and ensuring they don't collide and that they all stay within a certain distance from the central point. The focal point will be input to the satellites from the computer then the satellites will move into position around the point themselves without any other instruction from the computer.

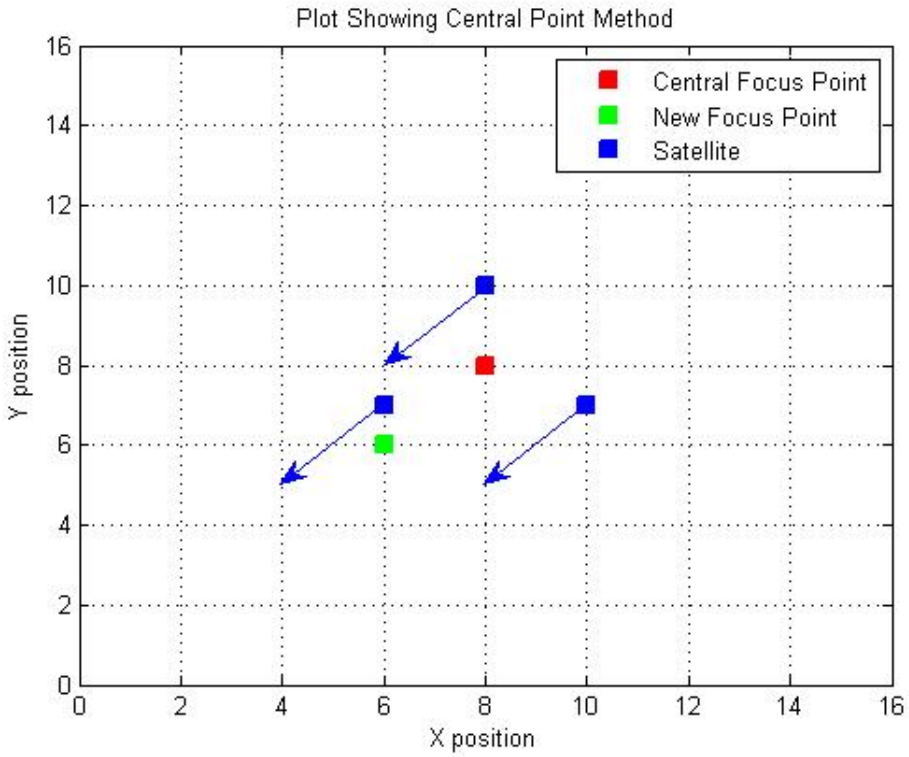

Figure 28: Simulation showing how the satellites move when the central focus changes

Figure 28 shows one method with the central point how all the satellites would take the most direct path to the correct position so the satellites should move the same distance as the focus point moves.

Alternatively the satellites could be programmed so that they must attempt to maintain a minimum distance in which case the arrow would curve out and away from the new point and move around to the new position (Schwartz and Hall 2003).

9.2.3 Safe Zone Movement

The chosen method of how the satellites would move was to use a system whereby the satellites will communicate their position to the other satellites. An individual satellite would then take this position and create an area around it which it is not allowed to move into, a 'safe zone'.

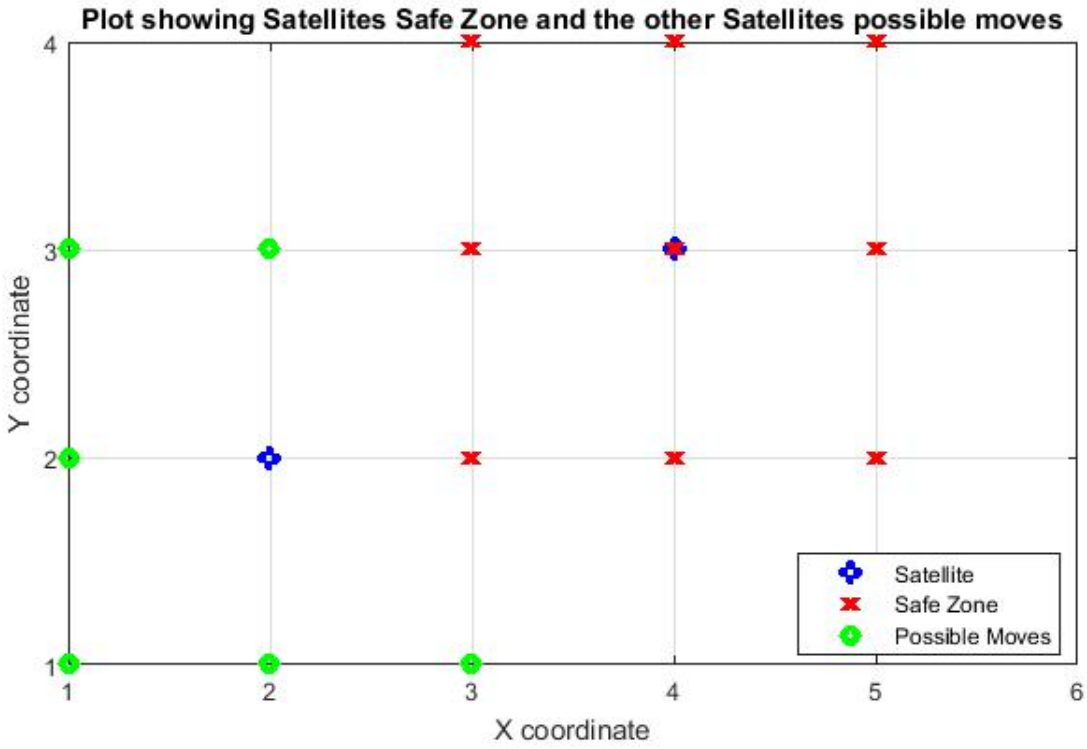

Figure 29: Simulation of possible moves for the left hand satellite based on the right hand satellite's current position

If another satellite is moving the safe zone for that satellite is increased to include the grid positions around the position that it is moving to. This is done to try to ensure that the satellites will not collide. The satellite will then compare the possible moves that it can make to those moves that are banned by the other satellites safe zones and then pick the next position which will mean that it progresses toward the orbiting point that it is aiming for.

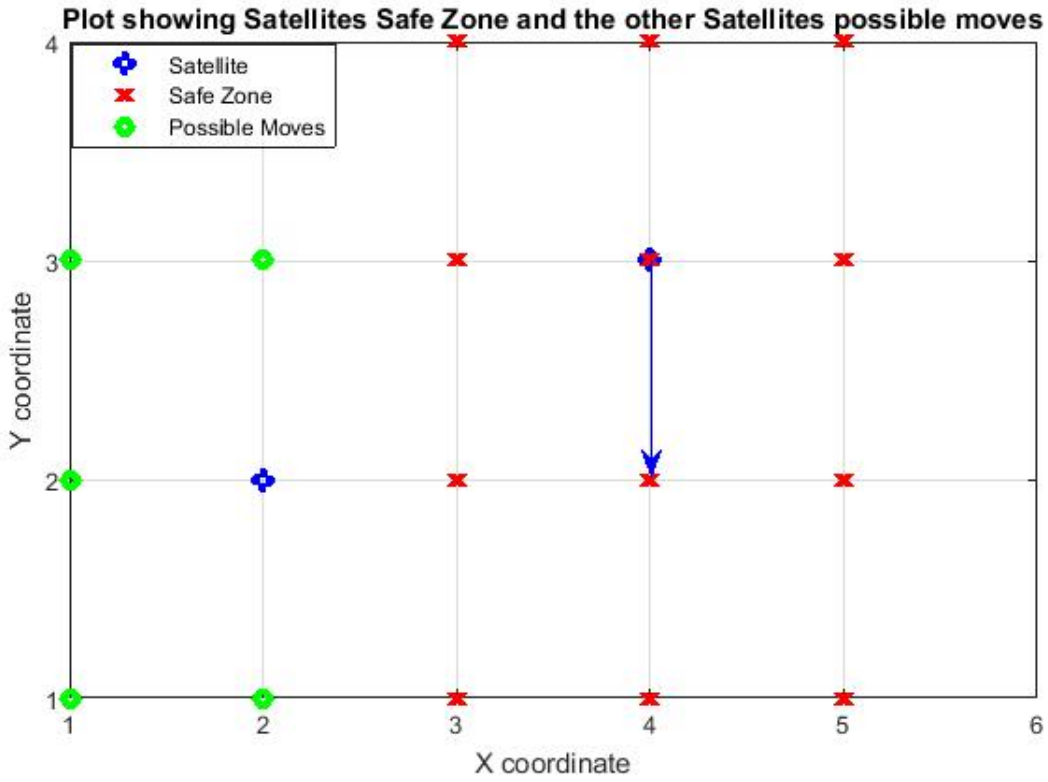

Figure 30: Simulation of possible moves for the left hand satellite when the right hand satellite is moving from its current position to (4,2)

Figure 30 shows that the satellite on the left can no longer move to position (3,1) as the right hand satellite is now moving to a new position (4,2).

This was decided with the idea of using two of the satellites on a 6x4 grid; if three satellites were to be used then the size of the grid would need to be increased otherwise the number of positions that would be banned by the other satellites would have a high potential to cause the satellites to sit still as no move that it could make would progress it toward the orbital point.

The movement to the next point would be using a pre-programmed function which will take its current position, the required position and the satellite's orientation. Based on these the correct motors would then be powered up so that it travels to the next location. The movement from one point to another would therefore be open loop control which is undesirable however this solution was selected for simplicity as it would be easier to implement. When the satellite reached the next position it would come to a stop before re evaluating the other satellite positions and planning its next move.

The satellites would make decisions based on a round robin approach where one makes a decision first, then once it has started moving and sent where it is going to the other satel-

lites the next satellite will then make a decision. This will continue to all the other satellites making a decision one at a time before looping back round to the original satellite. This may cause some of the satellites to delay slightly if the preceding satellite is moving diagonally as it will have a further distance to travel.

If a satellite is adjacent to the orbital point it would then proceed to 'orbit' the point in a clockwise fashion moving in a square from one grid position to the next around the orbital point. If both satellites are around the orbiting point then they would both move in a clockwise manner. This orbital method works with two satellites for continual movement however if there were three satellites then there would only be one satellite moving at a time due to the crossover of safe zones.

If a satellite finds that it is inside the safe zone of another satellite it will immediately shut down in order to try to prevent collision with the rotors spinning.

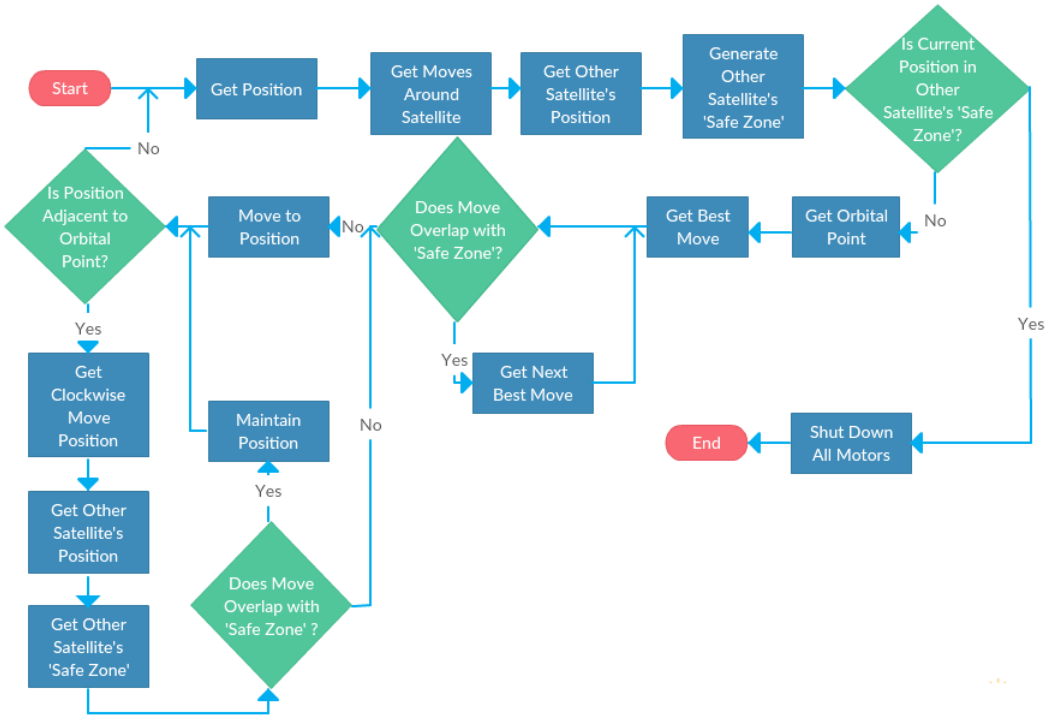

Figure 31: Flow Chart of the decision process for the movement of a satellite

10 System Integration

Despite the completion of the computer vision, control for 3 DoF and hardware the group was unable to integrate all of the completed sections. No work was done on the ODROID hardware itself. For the final demonstration a demo and test mode unit was constructed allowing each motor to be individually controlled by a human operator and for a wired failsafe. The computer vision was written using OpenCV in ROS so that is can easily be transferred onto the ODROID. The adaptable model allows for rapid prototyping, control algorithm design and simulation of different satellite configurations.

The work completed during this project has provided a strong base upon which future work can be done. The model allows for simulation and rapid comprehension of the system in both 3 and 5 DoF. The computer vision and grid system can be easily implemented on the ODROID. The hardware is now well documented and homogenous between satellites giving a good platform for testing and demonstration.

11 Conclusion

11.1 Future Effort

A controller suitable for implementation on the 5 DOF hardware will need to be designed and validated. Ideally a Graphical User Interface will be produced which is able to issue setpoint commands and demonstrate satellite dynamics in real time, making the model valuable as a promotional as well as a technical tool. For this the ODROID must be wirelessly connected to the desktop over Wi-Fi.
The computer vision and formation control algorithms must be adapted for the ODROID such that they can run simultaneously.

11.2 Summary of Achievements

The original hardware for this project has been replaced with homogeneous and well documented components. The low level code to control this hardware has been written such that 8 control inputs are translated to calibrated thrusts. Additionally a dedicated piece of test and demo hardware has been constructed allowing for hardware and software troubleshooting. The hardware is now easily programmable using only the onboard ODROID.
A functional and flexible model was developed in both 3 and 5 DOF, enabling the design of a 3 DOF control algorithm and providing a clear path for the development of a final 5 DOF hardware design and controller.

11.3 Major Changes to Project Plan

Integration not acheived

11.4 Summary

This report concerned three table top model satellites under construction for the purposes of space formation flying.

A detailed selection criterion for the components was ascertained from the project requirements; first the propulsion system was specified as this was the easiest to calculate from the requirements. The rest of the selection focused on compatibility. The thrust needed for the propulsion systems was approximated and two actuator placements were compared.

The propellers and motors were tested to check their viability, and it was found that running the slow fly propellers in reverse was not viable. The hardware was constructed and the low level code written with a strong focus on documentation and comprehensibility for future users.

The model allowed us to prove that satellites are controllable in 3DOF using PID control, the coefficients of which were found; however a more robust controller strategy will need to be implemented on the 5DOF model.

The computer vision can identify specific colours within an image and also draw contours around shapes. This was achieved via filtering of the image using a variety of different filtering techniques. Converting the image to HSV allowed the colour of the shapes to be identified; converting the image to binary enabled edge detection.

The method of localisation is to be a grid with a look-up table: the grid shall consist of multiple colours and shapes, with each colour identifying the y co-ordinate and the number of edges of the shape identifying the x co-ordinate. A method of satellite formation movement was identified, in which a safe zone area is created around the other satellites into which the satellite can not move; from the remaining positions available a position is chosen which moves it toward the orbital point.

Although integration of the system was not achieved, the hardware, computer vision subsystem and control algorithm have all separately been designed and tested, and implementation of the 3 DOF system should now be a relatively simple process.

12 References

Arduino (2016). *The Arduino Forum PWM Library Frequencies*. URL: https://www.arduino.cc/.

Argentim, Lucas M et al. (2013). "PID, LQR and LQR-PID on a quadcopter platform". In: *Informatics, Electronics & Vision (ICIEV), 2013 International Conference on*. IEEE, pp. 1–6.

Beeran Kutty, Suhaili et al. (2014). "Evaluation of canny and sobel operator for logo edge detection". In: *Technology Management and Emerging Technologies (ISTMET), 2014 International Symposium on*. IEEE, pp. 153–156.

Bradski, Gary and Adrian Kaehler (2008). *Learning OpenCV: Computer vision with the OpenCV library*. " O'Reilly Media, Inc."

Brandt, John B and Michael S Selig (2011). "Propeller performance data at low reynolds numbers". In: *49th AIAA aerospace sciences meeting*, pp. 2011–1255.

Chen, Teh-Chuan and Kuo-Liang Chung (2001). "An efficient randomized algorithm for detecting circles". In: *Computer Vision and Image Understanding* 83.2, pp. 172–191.

Class, Kim A and Randolph G Hartman (1996). *Look ahead satellite positioning system position error bound monitoring system*. US Patent 5,504,492.

Cui, Hai-Ying, Jun-Feng Li, and Yun-Feng Gao (2006). "An orbital design method for satellite formation flying". In: *Journal of mechanical science and technology* 20.2, pp. 177–184.

Dahl, PR (1968). *A solid friction model*. Tech. rep. DTIC Document.

Fernando, HCTE et al. (2013). "Modelling, simulation and implementation of a quadrotor UAV". In: *Industrial and Information Systems (ICIIS), 2013 8th IEEE International Conference on*. IEEE, pp. 207–212.

Foundation, Open Source Robotics (2016a). *ROS - Catkin Workspace*. URL: http://wiki.ros.org/catkin/workspaces.

— (2016b). *ROS - Nodes*. URL: http://wiki.ros.org/Nodes.

Garage, Willow (2016). *ROS - Willow Garage*. URL: https://www.willowgarage.com/pages/software/ros-platform.

Gedraite, Estevão S. and Murielle Hadad (2011). "Investigation on the effect of a Gaussian Blur in image filtering and segmentation". In: *Proceedings Elmar - International Symposium Electronics in Marine*, pp. 393–396.

Gordon (2016). *Why do some planes still use propeller engines, not jets?* URL: www.quora.com:%20https://www.quora.com/Why-do-%20some-planes-still-%20use-propeller-%20engines-not-%20jets.

Harath (2016). *OpenCv: QR Code Detection and Extraction*. URL: http://dsynflo.blogspot.co.uk/2014/10/opencv-qr-code-detection-and-extraction.html.

Hesselmar, Petra and Kristoffer Brodin (2015). "Formation Flying for Quadcopters". In:

Jirinec, Tomas (2011). "Stabilization and control of unmanned quadcopter". In: *Master's Thesis, Lulea University of Technology*.

Kwon, Daniel W (2010). "Propellantless formation flight applications using electromagnetic satellite formations". In: *Acta Astronautica* 67.9, pp. 1189–1201.

Lau, Chung (1995). *Rapid satellite signal acquisition in a satellite positioning system*. US Patent 5,418,538.

Lincoln, N. K. and S.M. Veres (2006). "Components of a vision assisted constrained autonomous satellite formation flying control system". In: *International Journal of Adaptive Control and Signal Processing*.

Mahony, Robert, Vijay Kumar, and Peter Corke (2012). "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor". In: *Robotics & Automation Magazine, IEEE* 19.3, pp. 20–32.

Maragos, Petros and Lúcio FC Pessoa (1999). "Morphological filtering for image enhancement and detection". In: *analysis* 13, p. 12.

McCormick, BW (1979). "The analysis of propellers including interaction effects". In:

McCurdy, David A (1985). "Comparison of advanced turboprop and conventional jet and propeller aircraft flyover noise annoyance: Preliminary results". In:

Meriam, James L and L Glenn Kraige (2012). *Engineering mechanics: dynamics*. Vol. 2. John Wiley & Sons.

Merrill, Garrick and Chris Becker (2015). "Formation Flying for Satellites and UAVs". In:

Mraz, Stephen J (2001). "INTERNATIONAL SPACE STATION UNDER CONSTRUCTION". In: *Machine Design* 73.14, p. 42.

Mu, Rongjun and Xin Zhang (2014). "Control allocation design of reaction control system for reusable launch vehicle". In: *Abstract and Applied Analysis*. Vol. 2014. Hindawi Publishing Corporation.

Ogawa, Kohei, Yasuaki Ito, and Koji Nakano (2010). "Efficient Canny edge detection using a GPU". In: *Networking and Computing (ICNC), 2010 First International Conference on*. IEEE, pp. 279–280.

OpenCV, Dev. Team (2016). *OpenCV 2.4.13.0 documentation - Eroding and Dilating*. URL: `http://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html`.

OpenCV3, Dev. Team (2016). *OpenCV 3.1.0 - Contours : Getting Started*. URL: `http://docs.opencv.org/3.1.0/d4/d73/tutorial_py_contours_begin.html#gsc.tab=0`.

Qu, Zhong et al. (2013). "An Improved Keyframe Extraction Method Based on HSV Colour Space". In: *Journal of Software* 8.7, pp. 1751–1758.

Rudow (2016). *Understanding Propeller Pitch*. URL: `http://www.boats.com/how-to/understanding-propeller-pitch/`.

Sabol, Chris, Rich Burns, and Craig A. McLaughlin (2001). "Satellite Formation Flying Design and Evolution". In: *Journal of Spacecraft and Rockets* 38.2, pp. 207–278.

Saska, Martin et al. (2014). "Coordination and navigation of heterogeneous MAV–UGV formations localized by a 'hawk-eye'-like approach under a model predictive control scheme". In: *The International Journal of Robotics Research*, p. 0278364914530482.

Schwartz, Jana L and Christopher D Hall (2003). "The distributed spacecraft attitude control system simulator: development, progress, plans". In: *NASA Space Flight Mechanics Symposium, Greenbelt, MD*. Citeseer.

Seong, Jae-Dong and Hae-Dong Kim (2013). "Optimization of Space Debris Collision Avoidance Maneuver for Formation Flying Satellites". In: *Journal of Astronomy and Space Sciences* 30.4, pp. 291–298.

Sheynblat, Leonid (2000). *Method and system for using altitude information in a satellite positioning system*. US Patent 6,061,018.

Sinha (2016). *Scanning QR Codes*. URL: http://aishack.in/tutorials/scanning-qr-codes-2/.

Spakovszky (2016). *Unified: Thermodynamics and Propulsion: Performance of Propellers*. URL: http://web.mit.edu/16.unified/www/FALL/thermodynamics/notes/node86.html.

Stabler (2016). *Inside Story*. URL: http://www.rcgroups.com/forums/showthread.php?t=185271&highlight=bergmeyer.

Staples (2016). *Propeller Static & Dynamic Thrust Calculation - Part 2 of 2 - How Did I Come Up With This Equation?* URL: http://www.electricrcaircraftguy.com/2014/04/propeller-static-dynamic-thrust-equation-background.html.

Stokes, George Gabriel (1851). *On the effect of the internal friction of fluids on the motion of pendulums*. Vol. 9. Pitt Press.

Systems, E-con. $e\text{-}CAM51_USB{-}5MPHDOEMUSBCameraModule$. URL: http://www.e-consystems.com/5mp-usb-cameraboard.asp.

Waltz, Frederick M and John WV Miller (1998). "Efficient algorithm for gaussian blur using finite-state machines". In: *Photonics East (ISAM, VVDC, IEMB)*. International Society for Optics and Photonics, pp. 334–341.

13 Appendix A - Updated Requirements

Requirements

1. Functional Requirements
    1. The three satellites shall be able to move in formation.
    2. Each satellite shall be semi-autonomous.
    3. The satellites shall move with 5 degrees of freedom.
        1. 3 degrees of rotational movement (roll, pitch and yaw).
        2. 2 degrees of translational movement (sway and surge).
    4. The satellites shall communicate with each other using a wireless network (WiFi).
    5. The system shall operate in a low friction environment.
    6. The satellites shall have sufficient sensors to detect its orientation and position.
        1. The satellites shall utilise computer vision techniques to locate itself relative to the table.
        2. The satellites shall detect its roll, pitch and yaw.
        3. The satellites shall locate one another to within 2-3 cm measured from the base.

2. Hardware
    1. The satellites' base shall use ball bearings to create the low friction interface between the table and the base.
    2. The satellites' frame shall allow the distribution of weight to be adjusted.
    3. The system shall have 8 actuators in the form of independently acting propellers - similar to RCS.
    4. Each satellite shall be equipped with two to four 5 megapixel cameras.
    5. Each satellite shall be equipped with a 3 axis gyroscope.
    6. Each satellite shall be independently powered using lithium polymer batteries.
    7. Each satellite shall use a microcontroller (Arduino UNO R3) for controlling the position and orientation of the satellites and communication.
    8. Each satellite shall use onboard computer Odroid XU3 4 core for image processing.
    9. The total weight of each satellite shall be around 8.5 kg (20 %).
    10. The total size of the satellites shall not exceed 540x540x500mm (WxLxH).

3. Performance
    1. Satellites shall move at no more than 1m/s.
    2. The satellites shall be able to move at a speed of 5 cm/s.
    3. The battery life of the satellites shall allow 5 minutes of continuous operation.

4. Software
    4.1 The software shall be written using object oriented techniques.

4.1.1 The code shall be written using an object oriented programming language namely C++ (with the Indigo Ros library) .

4.1.2 The code shall be modularised.

4.2 The code shall be written so as to be transparent to others, using comments and a readable structure.

4.3 The code shall make use of the OpenCV library for computer vision techniques (localisation/mapping).

4.4 The CPU shall run using a linux debian-based operating system (Ubuntu 14.04).

5. Interface/Communication

1. The system shall use wifi to communicate between satellites while operating in formation.
2. The system shall receive software updates over wifi from the connected PC.

6. Reliability

6.1 The satellites shall enter a safe mode in the case of system failure which shall cut power to all propellers, this safe mode shall be activated using a remote control to prevent users from having to touch the satellite when the rotors are spinning.

6.2 The satellites shall consistently complete a 5 minute demonstration without failure.

7. Environmental

1. The satellites shall operate on a glass table of dimensions 2.4 x 3.4m.
    1. The table shall have a low friction surface.
2. The table shall be wiped before every use to prevent dust from causing degradation of the ball bearing base.
3. The edges of the table shall be marked with coloured markers in order to make orientation and location mapping easier using computer vision.
4. There shall be a grid suspended above the table to allow the satellites to self localise by reading off position data from the grid.

8. Safety

1. Each satellite shall have a RC shutdown in case of emergency such as an imminent collision.
2. Programming shall not allow satellites to collide with each other.
3. Programming shall not allow satellites to collide with sides of table.
4. Table will have barriers around perimeter to prevent robots from falling.
5. Batteries will be used and charged according to manufacturer specifications.
    1. Batteries will operate within the manufacturer stated safe voltages and draw below the stated safe current.
    2. The batteries shall be fitted with a low voltage detection device to indicate when the batteries are getting below

their safe operating voltages.
6. Satellites shall not be touched by anyone while they are active.
7. Shall only operate on the glass table provided by the department.

9. Optional Requirements
    1. The system should demonstrate a range of formations such as 'triangular juggling'.
    2. The system should demonstrate object avoidance.
        1. The satellites should map out the locations of obstacles and share this information.
    3. The system should respond effectively to large scale disturbances that cause more than one satellite to need to react.
    4. The system should display a GUI on a connected PC, this shall display:
        1. A top down feed of satellite positions.
        2. A video feed from each satellite.

14 Appendix B - Arduino Code for ESC initialisation & Motor Control

```
// Theodore Abbott
// 120139831
// 25 / 02 / 2016
// Function to:
// Initialise and calibrate 8 ESCs
// Take input from 6 analogue pins and convert those signals to motor outputs
// Check every cycle that a safety button has not been released and stop all motors for 5 seconds if it has.

#include <Servo.h>                                      //used to setup servo objects
#include <PWM.h>                                        //used to set the PWM frequency

#define frequency 300                                   //used to hold the desired PWM frequency
#define deadzone 100                                    //used to create deadzone for analouge input
#define safetyPin 1                                     //used to hold the pin used for failsafe

bool safe = false;                                      //boolean value, when false the ESC's recieve an off signal, when true the ESC's recieve the desired signal
int i=0;                                                //simple incremental variable
int throttleMap[6] = {0,0,0,0,0,0};                     //array to hold the desired power value for each motor
int throttle[6] = {0,0,0,0,0,0};                        //array to hold the desired power value for each motor once they have been mapped over the motor opperating range
int pwmPins[6] = {3,5,6,9,10,11};                       //array to hold the PWM capable pins
static const uint8_t inputPins[6] = {A0,A1,A2,A3,A4,A5}; //used to hold the addresses of the 6 analog input pins

Servo escs[6];                                          //these ESCs are the PWM outputs of the arduino going to each motor, esc5 and esc6 control the 2 pairs of motors on the top of the satellite

void setup()                                            //Runs once on arduino powerup
{
  for(i=0;i<5;i++)                                      //set the analog pins as inputs
  {
    pinMode(inputPins[i], INPUT);
  }
```

```
31    pinMode(safetyPin, INPUT);                          //set the digital
          safety pin as an input
32    Serial.begin(9600);                                 //set up serial I/O
33
34    for(i=0;i<5;i++)
35    {
36      SetPinFrequencySafe(pwmPins[i],frequency);        //setting the PWM
            frequency of each of the PWM capable pins to the value selected
            by frequency
37      escs[i].attach(pwmPins[i]);                       //attaching each esc
            object to the approprate pin
38    }
39
40    for(i=0;i<5;i++)
41    {
42      escs[i].write(23);                                //write the zero
            throttle values the physical ESCs are expecting in order to
            initialise
43    }
44    delay(5000);                                        //gives the physical
          ESCs time to initialise before they start receiving control
          signals
45
46  }
47
48  void loop()                                           //loops until arduino
        shut down
49  {
50    for(i=0;i<5;i++)                                    //get the throttle
          values and map them to the throttle array
51    {
52      throttleMap[i] = analogRead(inputPins[i]);        //read the value
            coming in from the input pin array
53      if(throttleMap[i] > deadzone)                     //if the analogue
            signal is greater than the deadzone then
54      {
55        throttle[i] = map(throttleMap[i], 0, 1023, 23, 179);   //maps the
              throttle value from 0-1023 to 23-179 as this is what ESC takes
               as an input
56      }
57      else throttle[i] = 0;                             //else ignore the
            analogue input
58    }
59
60    safe = false;
61    if(digitalRead(safetyPin) == HIGH) safe = true;     //if the safety
          input is being held high then the device is safe
62
63    if(safe == true)                                    //if the failsafe
          hasnt been triggered
64    {
65      for(i=0;i<5;i++)
66      {
```

```
67        escs[i].write(throttle[i]);                    //write the
              throttle values
68      }
69      Serial.println("Failsafe not triggered.");
70    }
71
72    else
73    {
74      escs[0].write(0);
75      escs[1].write(0);
76      escs[2].write(0);
77      escs[3].write(0);
78      escs[4].write(0);
79      escs[5].write(0);
80      Serial.println("Failsafe triggered, please wait 5 seconds.");
81      delay(5000);
82    }
83  }
```

15 Appendix C - Catkin Workspace - CMakeLists.txt

```
cmake_minimum_required(VERSION 2.8.3)
project(BroadcastWebcam)

## Find catkin macros and libraries
## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
## is used, also find other catkin packages
find_package(catkin REQUIRED COMPONENTS roscpp rospy std_msgs cv_bridge
   sensor_msgs image_transport message_generation image_geometry)
find_package(OpenCV 2 REQUIRED)

## System dependencies are found with CMake conventions
# find_package(Boost REQUIRED COMPONENTS system)

## Uncomment this if the package has a setup.py. This macro ensures
## modules and global scripts declared therein get installed
## See http://ros.org/doc/groovy/api/catkin/html/user_guide/setup_dot_py
   .html
# catkin_python_setup()

######################################
## Declare ROS messages and services ##
######################################

# Generate added messages and services with any dependencies listed here

#add_message_files(
#   FILES
#)

###################################
## catkin specific configuration ##
###################################
## The catkin_package macro generates cmake configure files for your
   package
## Declare things to be passed to dependent projects
## LIBRARIES: libraries you create in this project that dependent
   projects also need
## CATKIN_DEPENDS: catkin_packages dependent projects also need
## DEPENDS: system dependencies of this project that dependent projects
   also need
catkin_package(
   INCLUDE_DIRS include
#  LIBRARIES beginner_tutorials
   CATKIN_DEPENDS roscpp rospy std_msgs cv_bridge sensor_msgs
      image_transport
#  DEPENDS system_lib
)

###########
## Build ##
```

```
45 ###########
46
47 ## Specify additional locations of header files
48 ## Your package locations should be listed before other locations
49 include_directories(include
50   ${catkin_INCLUDE_DIRS}
51   ${OpenCV_INCLUDE_DIRS}
52 )
53
54 ## Declare a cpp library
55 # add_library(beginner_tutorials
56 #   src/${PROJECT_NAME}/beginner_tutorials.cpp
57 # )
58
59 ## Declare a cpp executable
60 # add_executable(beginner_tutorials_node src/beginner_tutorials_node.cpp
       )
61
62 ## Add cmake target dependencies of the executable/library
63 ## as an example, message headers may need to be generated before nodes
64 # add_dependencies(beginner_tutorials_node
       beginner_tutorials_generate_messages_cpp)
65
66 ## Specify libraries to link a library or executable target against
67 # target_link_libraries(beginner_tutorials_node
68 #   ${catkin_LIBRARIES}
69 # )
70
71 #############
72 ## Install ##
73 #############
74
75 # all install targets should use catkin DESTINATION variables
76 # See http://ros.org/doc/groovy/api/catkin/html/adv_user_guide/variables
       .html
77
78 ## Mark executable scripts (Python etc.) for installation
79 ## in contrast to setup.py, you can choose the destination
80 # install(PROGRAMS
81 #   scripts/my_python_script
82 #   DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
83 # )
84
85 ## Mark executables and/or libraries for installation
86 # install(TARGETS beginner_tutorials beginner_tutorials_node
87 #   ARCHIVE DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
88 #   LIBRARY DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION}
89 #   RUNTIME DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION}
90 # )
91
92 ## Mark cpp header files for installation
93 # install(DIRECTORY include/${PROJECT_NAME}/
94 #   DESTINATION ${CATKIN_PACKAGE_INCLUDE_DESTINATION}
```

```
#   FILES_MATCHING PATTERN "*.h"
#   PATTERN ".svn" EXCLUDE
# )

## Mark other files for installation (e.g. launch and bag files, etc.)
# install(FILES
#   # myfile1
#   # myfile2
#   DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION}
# )

#############
## Testing ##
#############

## Add gtest based cpp test target and link libraries
# catkin_add_gtest(${PROJECT_NAME}-test test/test_beginner_tutorials.cpp
   )
# if(TARGET ${PROJECT_NAME}-test)
#   target_link_libraries(${PROJECT_NAME}-test ${PROJECT_NAME})
# endif()

## Add folders to be run by python nosetests
# catkin_add_nosetests(test)


add_executable(BroadcastWebcam src/BroadcastWebcam.cpp)
target_link_libraries(BroadcastWebcam ${catkin_LIBRARIES} ${
   OpenCV_LIBRARIES})
```

16 Appendix D - Catkin Workspace -package.xml

```
<?xml version="1.0"?>
<package>
  <name>BroadcastWebcam</name>
  <version>0.0.0</version>
  <description>The BroadcastWebcam package</description>

  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>

  <!-- Url tags are optional, but mutiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://ros.org/wiki/beginner_tutorials</url> -->

  <!-- Author tags are optional, mutiple are allowed, one per tag -->
  <!-- Authors do not have to be maintianers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->

  <!-- The *_depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
  <!-- Use build_depend for packages you need at compile time: -->
  <!--   <build_depend>message_generation</build_depend> -->
  <!-- Use buildtool_depend for build tool packages: -->
  <!--   <buildtool_depend>catkin</buildtool_depend> -->
  <!-- Use run_depend for packages you need at runtime: -->
  <!--   <run_depend>message_runtime</run_depend> -->
  <!-- Use test_depend for packages you need only for testing: -->
  <!--   <test_depend>gtest</test_depend> -->
  <buildtool_depend>catkin</buildtool_depend>
  <build_depend>roscpp</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>cv_bridge</build_depend>
  <build_depend>sensor_msgs</build_depend>
  <build_depend>std_msgs</build_depend>
  <build_depend>message_generation</build_depend>
  <build_depend>image_transport</build_depend>
  <build_depend>image_geometry</build_depend>
  <build_depend>OpenCV</build_depend>
  <run_depend>message_runtime</run_depend>
  <run_depend>roscpp</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>std_msgs</run_depend>
```

```
47    <run_depend>cv_bridge</run_depend>
48    <run_depend>sensor_msgs</run_depend>
49    <run_depend>image_transport</run_depend>
50    <run_depend>image_geometry</run_depend>
51    <run_depend>OpenCV</run_depend>
52
53    <!-- The export tag contains other, unspecified, tags -->
54    <export>
55      <!-- You can specify that this package is a metapackage here: -->
56      <!-- <metapackage/> -->
57
58      <!-- Other tools can request additional information be placed here
          -->
59
60    </export>
61  </package>
```

17 Appendix E - ROS Camera Subscriber/Publisher System

```
/* Code that sets up publisher/subscriber system with camera & pc
   Modified by: Dr. Aitken, Shiv Gangapersad */
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
using namespace cv;

class BroadcastWebcam
{
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Publisher image_pub_;

public:
  BroadcastWebcam()
    : it_(nh_)
  {
    ROS_INFO("Hello");

    image_pub_ = it_.advertise("/webcam_video/output_video", 1);
    processImages();

  }

  ~BroadcastWebcam()
  {
  }

  void processImages() {
    ros::Rate loop_rate(20);
    ROS_INFO("0");
    VideoCapture cap(0); // open the video camera no. 0

    ROS_INFO("1");

    if (!cap.isOpened())  // if not success, exit program
    {
    ROS_INFO("Webcam not opened");
        return;
    }

    //double dWidth = cap.get(CV_CAP_PROP_FRAME_WIDTH); //get the width
```

```
        of frames of the video
    //double dHeight = cap.get(CV_CAP_PROP_FRAME_HEIGHT); //get the
        height of frames of the video

    while (1)
    {
        Mat frame;
        bool bSuccess = cap.read(frame); // read a new frame from video

    if (frame.empty()) //When it finish the video it breaks out of the
        cycle
        {
            ROS_INFO("EMPTY_Frame");
        }

    if (!bSuccess) //if not success, break loop
    {
        ROS_INFO("Cannot read a frame from video stream");
        break;
    }
    else {

        // insert img processing code here //
        imshow("Image", frame);
        cv::waitKey(3);
    }
    ros::spinOnce();
    loop_rate.sleep();
    }
    return;
  }


};

int main(int argc, char** argv)
{
  ros::init(argc, argv, "ic_test");
  BroadcastWebcam bw;
  ros::spin();
  return 0;
}
```

18 Appendix F - Basic HSV Thresholding Code

```
/* Authors: Shiv Gangapersad
            Matthew Joy */
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdio.h>
#include <stdlib.h>
//#include <iostream>
using namespace std;
using namespace cv;

class BroadcastWebcam
{
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Publisher image_pub_;

public:
  BroadcastWebcam()
    : it_(nh_)
  {
    ROS_INFO("Hello");

    image_pub_ = it_.advertise("/webcam_video/output_video", 1);
    processImages();
  }
  ~BroadcastWebcam()
  {
  }

  void processImages() {
    ros::Rate loop_rate(20);
    ROS_INFO("0");
    VideoCapture cap(0); // open the video camera no. 0

    ROS_INFO("1");

    if (!cap.isOpened())  // if not success, exit program
    {
    ROS_INFO("Webcam not opened");
        return;
    }
    namedWindow("Control", CV_WINDOW_AUTOSIZE); //create a window called
        "Control"

    int iLowH = 0;
    int iHighH = 179;
```

```
50
51      int iLowS = 0;
52      int iHighS = 255;
53
54      int iLowV = 0;
55      int iHighV = 255;
56
57      //create trackbars in "control" window
58      cvCreateTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
59      cvCreateTrackbar("HighH", "Control", &iHighH, 179);
60
61      cvCreateTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 -
            255)
62      cvCreateTrackbar("HighS", "Control", &iHighS, 255);
63
64      cvCreateTrackbar("LowV", "Control", &iLowV, 255); //Value (0 - 255)
65      cvCreateTrackbar("HighV", "Control", &iHighV, 255);
66
67      while (1)
68      {
69          Mat frame;
70          bool bSuccess = cap.read(frame); // read a new frame from video
71
72      if (frame.empty()) //When it finish the video it breaks out of the
            cycle
73          {
74              ROS_INFO("EMPTY_Frame");
75          }
76
77      if (!bSuccess) //if not success, break loop
78      {
79          ROS_INFO("Cannot read a frame from video stream");
80          break;
81      }
82      else {
83          // insert img processing code here //
84          Mat imgHSV;
85          //Convert the captured frame from BGR to HSV
86          cvtColor(frame, imgHSV, COLOR_BGR2HSV);
87          Mat imgThresholded;
88          inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH,
                iHighS, iHighV), imgThresholded); //Threshold the image
89
90          imshow("Thresholded Image", imgThresholded);
91          imshow("Image", frame);
92          cv::waitKey(3);
93      }
94      ros::spinOnce();
95      loop_rate.sleep();
96      }
97      return;
98    }
99  };
```

```
100
101 int main(int argc, char** argv)
102 {
103   ros::init(argc, argv, "ic_test");
104   BroadcastWebcam bw;
105   ros::spin();
106   return 0;
107 }
```

19 Appendix G - HSV Thresholding Code with Gaussian Blur & Morphological Operations

```
/* Authors: Shiv Gangapersad
            Matthew Joy */
#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdio.h>
#include <stdlib.h>
//#include <iostream>
using namespace std;
using namespace cv;

class BroadcastWebcam
{
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Publisher image_pub_;

public:
  BroadcastWebcam()
    : it_(nh_)
  {
    ROS_INFO("Hello");

    image_pub_ = it_.advertise("/webcam_video/output_video", 1);
    processImages();

  }

  ~BroadcastWebcam()
  {
  }

  void processImages() {
    ros::Rate loop_rate(20);
    ROS_INFO("0");
    VideoCapture cap(0); // open the video camera no. 0

    ROS_INFO("1");

    if (!cap.isOpened())  // if not success, exit program
    {
    ROS_INFO("Webcam not opened");
        return;
    }
    namedWindow("Control", CV_WINDOW_AUTOSIZE); //create a window called
        "Control"
```

```
49
50      int iLowH = 0;
51      int iHighH = 179;
52
53      int iLowS = 0;
54      int iHighS = 255;
55
56      int iLowV = 0;
57      int iHighV = 255;
58
59      //create trackbars in "control" window
60      cvCreateTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
61      cvCreateTrackbar("HighH", "Control", &iHighH, 179);
62
63      cvCreateTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 -
            255)
64      cvCreateTrackbar("HighS", "Control", &iHighS, 255);
65
66      cvCreateTrackbar("LowV", "Control", &iLowV, 255); //Value (0 - 255)
67      cvCreateTrackbar("HighV", "Control", &iHighV, 255);
68
69      while (1)
70      {
71          Mat frame;
72          bool bSuccess = cap.read(frame); // read a new frame from video
73
74      if (frame.empty()) //When it finish the video it breaks out of the
            cycle
75          {
76              ROS_INFO("EMPTY_Frame");
77          }
78
79      if (!bSuccess) //if not success, break loop
80      {
81          ROS_INFO("Cannot read a frame from video stream");
82          break;
83      }
84      else {
85
86          // insert img processing code here //
87          Mat imgHSV;
88          //Convert the captured frame from BGR to HSV
89          cvtColor(frame, imgHSV, COLOR_BGR2HSV);
90          Mat imgThresholded;
91          inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH,
                iHighS, iHighV), imgThresholded); //Threshold the image
92          //morphological opening (remove small objects from the
                foreground)
93          erode(imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
94          dilate( imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
95
```

```
96          //morphological closing (fill small holes in the foreground)
97          dilate( imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
98          erode(imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
99
100         imshow("Thresholded Image", imgThresholded);
101         imshow("Image", frame);
102         cv::waitKey(3);
103     }
104     ros::spinOnce();
105     loop_rate.sleep();
106     }
107     return;
108   }
109
110 };
111
112 int main(int argc, char** argv)
113 {
114   ros::init(argc, argv, "ic_test");
115   BroadcastWebcam bw;
116   ros::spin();
117   return 0;
118 }
```

20 Appendix H - Matlab Script for Edge Detection

```
% Author Matthew Joy
clear
camera=webcam(2);
clear
%Open the Camera
camera=webcam(2);
%Only analysing one frame
for idx=1:1
    figure(idx)
    %Take Picture
    img=snapshot(camera);
    %Convert image to greyscale
    grey=rgb2gray(img);
    %Apply Sobel Filter
    BW=edge(grey,'Sobel',0.01);
    figure(1)
    imshow(BW)
    %Apply Canny Filter
    BW2=edge(grey,'Canny',0.1);
    figure(2)
    imshow(BW2)
    %Show Original Image
    figure(3)
    imshow(img)
end
%Close the camera
clear camera;
```

21 Appendix I - Final Computer Vision Code

```
/* Authors: Shiv Gangapersad
            Matthew Joy
    HSV Thresholding
    Contour Detection and Drawing */

#include <ros/ros.h>
#include <image_transport/image_transport.h>
#include <cv_bridge/cv_bridge.h>
#include <sensor_msgs/image_encodings.h>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <stdio.h>
#include <stdlib.h>
using namespace std;
using namespace cv;

// opens ROS Nodes to communicate with camera
class BroadcastWebcam
{
  // linkinf node handling and opencv libraries
  ros::NodeHandle nh_;
  image_transport::ImageTransport it_;
  image_transport::Publisher image_pub_;

public:
  BroadcastWebcam()
    : it_(nh_)
  {
    ROS_INFO("Hello");
    image_pub_ = it_.advertise("/webcam_video/output_video", 1);
    processImages();
  }

  ~BroadcastWebcam()
  {
  }
// Main image proc function
  void processImages() {
    ros::Rate loop_rate(20);
    ROS_INFO("0");
    VideoCapture cap(0); // open the video camera no. 0
    ROS_INFO("1");
    // checks if camera is open or plugged in
    if (!cap.isOpened())  // if not success, exit program
    {
    ROS_INFO("Camera not opened/Not plugged in");
        return;
    }
    //create a window called "Control" for HSV
    namedWindow("Control", CV_WINDOW_AUTOSIZE);
```

```
51      //Hue
52      int iLowH = 0;
53      int iHighH = 179;
54      //Saturation
55      int iLowS = 100;
56      int iHighS = 255;
57      //Value of Luminosity
58      int iLowV = 0;
59      int iHighV = 255;
60
61      //Hue basic colour ranges - mainly for reference
62      //Orange
63      int OrangeLowH=0;
64      int OrangeHighH=22;
65      //Yellow
66      int YellowLowH=22;
67      int YellowHighH=38;
68      //Green
69      int GreenLowH=38;
70      int GreenHighH=75;
71      //Blue
72      int BlueLowH=75;
73      int BlueHighH=130;
74      //Violet
75      int VioletLowH=130;
76      int VioletHighH=160;
77      //Red
78      int RedLowH=160;
79      int RedHighH=179;
80
81      //create trackbars in "control" window
82      cvCreateTrackbar("LowH", "Control", &iLowH, 179); //Hue (0 - 179)
83      cvCreateTrackbar("HighH", "Control", &iHighH, 179);
84
85      cvCreateTrackbar("LowS", "Control", &iLowS, 255); //Saturation (0 -
            255)
86      cvCreateTrackbar("HighS", "Control", &iHighS, 255);
87
88      cvCreateTrackbar("LowV", "Control", &iLowV, 255); //Value (0 - 255)
89      cvCreateTrackbar("HighV", "Control", &iHighV, 255);
90
91      while (1)
92      {
93          Mat frame;
94          bool bSuccess = cap.read(frame); // read a new frame from video
95
96      if (frame.empty()) //When it finish the video it breaks out of the
            cycle
97          {
98              ROS_INFO("EMPTY_Frame");
99          }
100
101     if (!bSuccess) //if not success, break loop
```

```
102     {
103         ROS_INFO("Cannot read a frame from video stream");
104         break;
105     }
106     else {
107         Mat imgHSV;
108         //Convert the captured frame from BGR to HSV
109         cvtColor(frame, imgHSV, COLOR_BGR2HSV);
110         Mat imgThresholded;
111         //Filter out the colours specified outside the trackbar values
112         inRange(imgHSV, Scalar(iLowH, iLowS, iLowV), Scalar(iHighH,
                iHighS, iHighV), imgThresholded); //Threshold the image
113         // apply Gaussian blur filter to soften image
114         GaussianBlur(imgThresholded, imgThresholded, Size(5, 5), 0 ,0);
115 
116         //morphological opening (remove small objects from the
                foreground)
117         erode(imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
118         dilate( imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
119 
120         //morphological closing (fill small holes in the foreground)
121         dilate( imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
122         erode(imgThresholded, imgThresholded, getStructuringElement(
                MORPH_ELLIPSE, Size(5, 5)) );
123 
124         // This section below deals with contour/edge detection
125         int thresh = 100;
126         vector<vector<Point> > contours;
127         Mat imgBin;
128         vector<Vec4i> hierarchy;
129 
130         // Detect edges using Threshold
131         threshold( imgThresholded, imgBin, thresh, 255, THRESH_BINARY );
132 
133         //finding all contours in the image
134         findContours(imgBin, contours, hierarchy, CV_RETR_CCOMP,
                CV_CHAIN_APPROX_SIMPLE, Point(0,0));
135 
136         vector<vector<Point> > contours_poly( contours.size() );
137         int result;
138         int i;
139 
140         for(i=0; i< contours.size(); i++){
141             //obtain a sequence of points of contour
142             approxPolyDP(Mat(contours[i]), contours_poly[i], 1, true);
143         }
144 
145         RNG rng(12345);
146         Mat drawing=frame.clone();
147         for(i = 0; i< contours.size(); i++ )
```

```
148         {
149             Scalar color = Scalar( rng.uniform(0, 255), rng.uniform
                (0,255), rng.uniform(0,255) );
150             //Draw contours around shape
151             drawContours( drawing, contours, i, color, 2, 8, hierarchy,
                0, Point() );
152         }
153         //Print the number of contours to screen
154         ROS_INFO("%d", i);
155 
156         /// Show in a window
157         namedWindow( "Contours", CV_WINDOW_AUTOSIZE );
158         //Show Image with Contour lines drawn around shapes
159         imshow( "Contours", drawing );
160         //Show HSV image
161         imshow("Thresholded Image", imgThresholded);
162         cv::waitKey(3);
163     }
164     //Ros node operations
165     ros::spinOnce();
166     loop_rate.sleep();
167     }
168     return;
169   }
170 
171 };
172 // main program that runs Ros nodes and img proc function
173 int main(int argc, char** argv)
174 {
175   ros::init(argc, argv, "ic_test");
176   BroadcastWebcam bw;
177   ros::spin();
178   return 0;
179 }
```

22 Appendix J - Simulink Model Structure

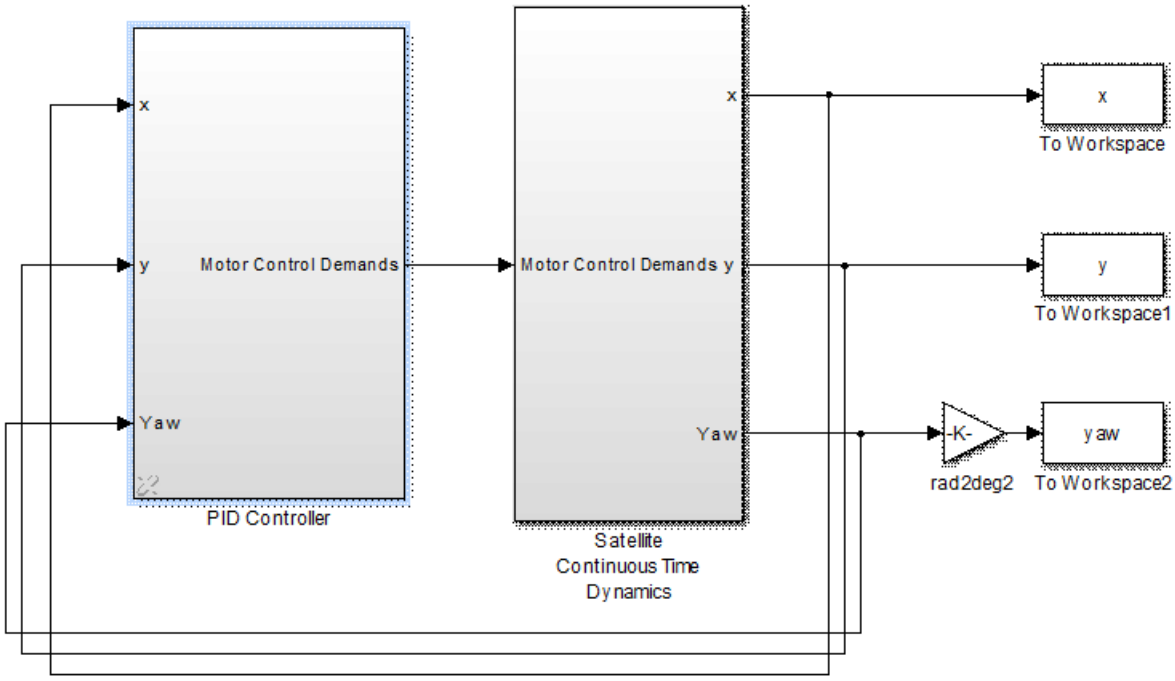

Figure 32: Simulink block representation showing high level closed loop feedback

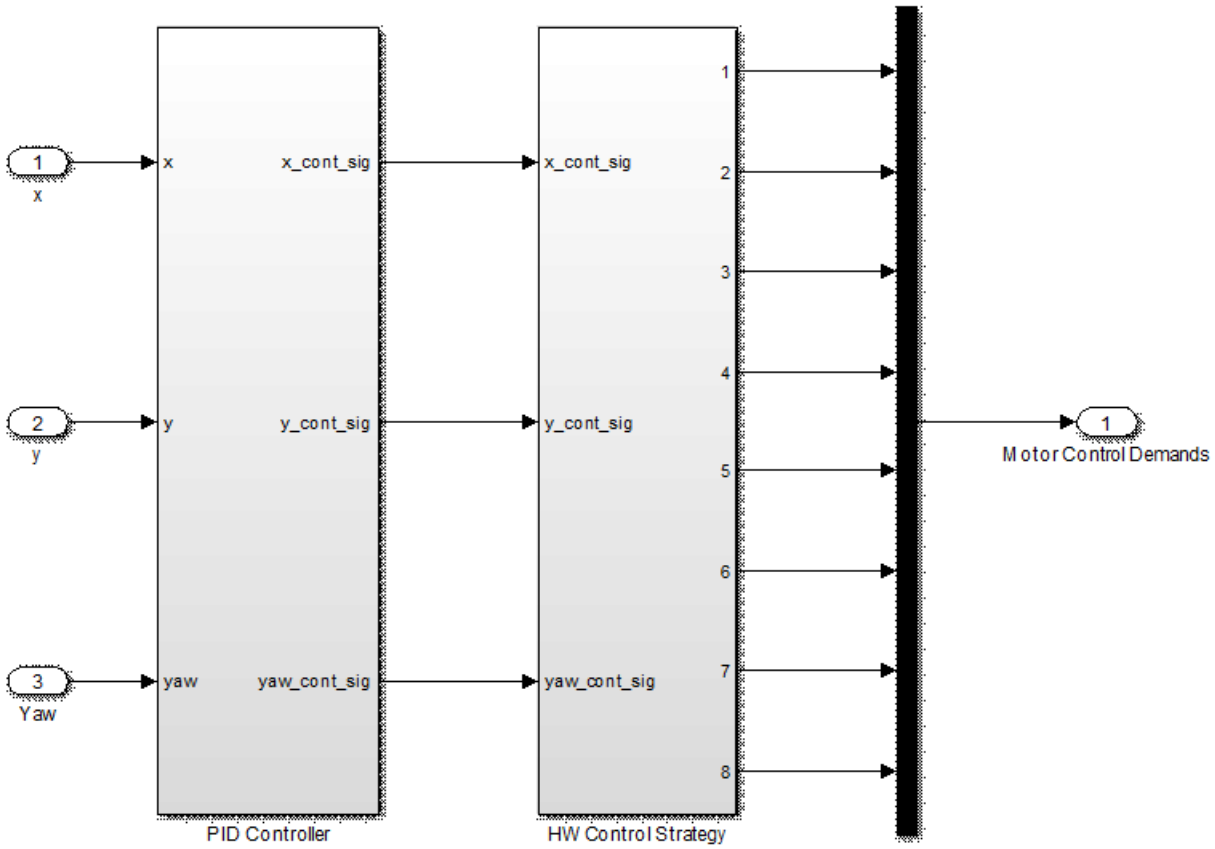

Figure 33: Simulink block representation showing the modelling of the control modelling strategy, separating the hardware control strategy from the control algorithm - Expansion of "PID Controller" block

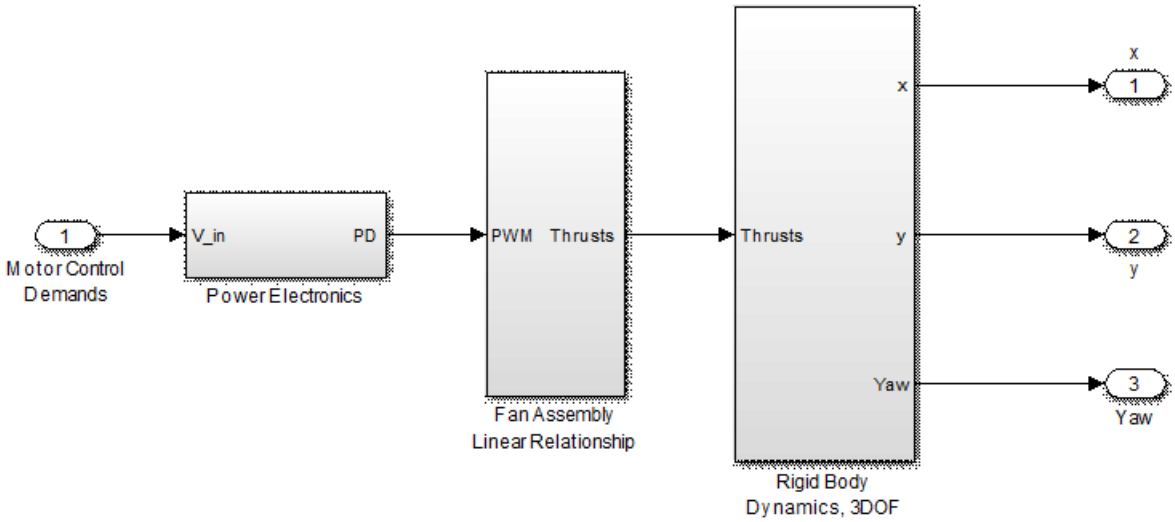

Figure 34: Simulink block representation showing the satellite continuous time dynamics, consisting of the power electronic dynamics, fan assembly linear relationship, (section 6.2) and the satellite rigid body dynamics, (section 6.3) - Expansion of "Satellite Continuous Time Dynamics" block

23 Appendix K - List of Parts

Component Type	Component	Individual Price / £	Number Needed	Price / £
Processor	ODROID UX3	0	3	0
Microcontroller	Arduino Uno R3	0	3	0
Sensor	Arduino 9 Axes Sensor Shield	15.89	3	47.67
Propeller	5x TOWER PRO 10x4.7SF	3.4	8	27.2
ESC card	Overlander XP2 ESC Program Card	7.2	1	7.2
Speed Controller	Overlander XP2 QUAD 20A Brushless ESC	30	6	180
Motor	Tornado Thumper V3 3536/08 1050KV Brushless Outrunner	14.4	26	374.4
Battery	2200mAh 3S 11.1v 30C LiPo Battery	12	6	72

Figure 35: List of Parts

24 Appendix L - Thrust Characteristic Test Results

Power Demand	Forward Thrust (g)	Reversing Thrust (g)	Lost (g)	Efficiency of running in reverse		
0	0	0	0	0		
2.5	70	0	70	0.00%		
5	136	0	136	0.00%		
7.5	170	0	170	0.00%		
10	270	38	232	14.07%		
12.5	310	40	270	12.90%		
15	401	47	354	11.72%		
17.5	490	51	439	10.41%		
20	527	52	475	9.87%		
22.5	630	64	566	10.16%		
25	756	78	678	10.32%		
27.5	844	100	744	11.85%		
30	959	130	829	13.56%		
32.5	1050	150	900	14.29%		
35	1059	201	858	18.98%		
37.5	1111	300	811	27.00%		
40	1152	320	832	27.78%		
42.5	1160	337	823	29.05%		
45	1148	340	808	29.62%		
47.5	1135	335	800	29.52%		
50	1130	340	790	30.09%		
Average	25.00	690.86	139.19	551.67	15.56%	

Figure 36: Thrust Characteristic Test Results