Spring 2017

Hovercam

Kevin Rauh
kcr23@zips.uakron.edu

Ross Palenik
rap78@zips.uakron.edu

Hovercam

Ross Palenik, Kevin Rauh

Department of Electrical Engineering

Honors Research Project

Submitted to

The Honors College

Approved:

Date 4/27/17
Honors Project Sponsor (signed)

HAMID BAHRAMI
Honors Project Sponsor (printed)

Date 04/27/17
Reader (signed)

Kye-Shin Lee
Reader (printed)

Date 4/28/17
Reader (signed)

Gregory A. Lewis
Reader (printed)

Accepted:

Date 4/28/17
Department Head (signed)

Joan Carletta
Department Head (printed)

Date 1 May 2017
Honors Faculty Advisor (signed)

ROBERT J. VEILLETTE
Honors Faculty Advisor (printed)

Date _____
Dean, Honors College

Senior Design/Honors Research Project Contribution

Ross Palenik

Electrical Engineer

2623037

My senior design/honors research project was the Hovercam. It involved the construction of an autonomous quadcopter that tracks, follows, and records a user holding the smart phone running the Hovercam mobile application. I was the team leader in my design group. My responsibilities ranged from assigning tasks to organizing our design schedule. I gave each of my teammates their core task as well as taking on the development of the mobile application. I was one of the group members responsible for writing the introduction paragraphs of the report along with the background section that pertains to the application. We all contributed to the marketing and design requirement sections. I, also, wrote the sections involving the design of our mobile app and constructed the flowcharts involving the application. The Gantt charts were my responsibility as team leader as well. I had a hand in the overall editing too. The construction of the actual mobile app was my main concern after assigning all of the group members' tasks. Once the application reached a stopping point, I took charge of assembling our final presentation PowerPoint and video slides. During the demonstration day, I discussed the project with any faculty or interested parties that approached our table. The project was, overall, a good experience and I am glad I was involved.

Senior Design/Honors Research Project Contribution

Kevin Rauh

Electrical Engineer

2645340

My senior design project was the Hovercam. On my team I was responsible for the documentation and assembling of the report. I was also responsible for the flight controller on the Hovercam project. I calibrated all the sensors that are used on the Hovercam and integrated them with a control software to maintain stable flight I also designed a 5V LDO, a voltage regulator to maintain 5V going into the flight and the navigation controller. After designing the LDO I also developed a 5V switching regulator circuit that would have the same function as the LDO but would consume much less power. After the regulator, I assembled the quadcopter and connected the motors, Electronic speed controllers (ESC) so that we could begin testing. Working with our Software Engineer we could successfully communicate, using MavLink commands, between the Raspberry Pi navigation controller and the Pixhawk Flight controller. After communication was established I integrated the controllers into the quadcopter hardware so that they would be secure in flight and correctly wired together. I also wrote preliminary code for ultrasonic sensors that were to be used for obstacle avoidance while in flight. After the project was designed, I was the primary team member responsible for setting up the Hovercam for test flights.

Hovercam

Senior Project Final Report

Design team 12

Adam Fada (Hardware Lead)

Daniel O'Brien (Software Lead)

Ross Palenik (Team Leader)

Kevin Rauh (Archivist)

Faculty Advisor: Dr. Hamid Bahrami

Senior Design Coordinator: Gregory A. Lewis

May 1, 2017

1. Table of Contents+----

2. List of Figures

3. List of Tables

4. Abstract

Quadcopters are widely used in recreational areas and often have video cameras mounted to them. The Hovercam increases the range of usefulness of quadcopters and allow the user to record him or herself while performing any task. The Hovercam will use GPS to track and follow a target. A smartphone application will be used to control the Hovercam in its basic functions of taking off, hovering, and landing.

5. Problem Statement

A. Need

This product involves a camera mounted on an automated aerial device that will track a target from a user specified distance. A quadcopter will be used initially. The Hovercam will allow the user to record themselves when they are on the go. Some uses for this can be skiing, mountain biking, and four wheeling.

B. Objective

The objective is to construct and program an aerial device that will autonomously follow and film a target. The aerial device will be programmed to maintain a set distance from the target and the ground while avoiding obstacles. The aerial device will visually record the target. A smart phone application will send commands to take-off, land, hover, initiate autonomous tracking, and view live video feed.

C. Background

Frame

The Hovercam is built using the DJI F450 frame kit. The frame kit includes the following items:

1 x F450 frame kit
4 x Sunnysky X2212 980KV Brushless motors
4 x HP SimonK 30A Speed Controllers
2 x 1045(CW+CCW) Black Propellers
2 x 1045(CW+CCW) Red Propellers

Each item in the kit is researched and verified that it will work to meet the design requirements of the Hovercam. Figure 1 below displays the assembled frame kit for the Hovercam.

Figure 1: Hovercam Frame Kit

Implementing ultra-strength material, integrated PCB wiring, and assembly space with retractable arms the DJI F450 frame is durable and lightweight frame, meeting the specifications set in the marketing and design requirements. The material that the frame is constructed from is PA66+30GF which is an ultra-strength nylon polymer material. Table 1 below displays the material specifications of the frame.

Table 1: Frame Kit Specifications

Impact Strength, Izod	English	SI Metric
notched 1/8 in (3.2 mm) section	1.4 ft-lbs/in	75 J/m
unnotched 1/8 in (3.2 mm) section	13.0 ft-lbs/in	694 J/m
Tensile Strength	21500 psi	148 MPa
Tensile Elongation	2.5 - 3.5 %	2.5 - 3.5 %
Tensile Modulus	1.35 x 10^6 psi	9308 MPa
Flexural Strength	35000 psi	241MPa
Flexural Modulus	1.30 x 10^6 psi	8964 MPa

PCB wiring in the frame is implemented through a process of point-to-point wiring allowing for easy connections to various components. This allows the integration of new components such as controllers and sensors. Having a two deck center of the frame is very beneficial because it allows one to implement various flight controllers and multiple batteries.

Motor/Thrust Calculations:

The thrust output of a quadcopter must be greater than the weight of the quadcopter for it to fly. Each of the four motor that are used on a quadcopter provide a specific maximum thrust to the system. Multiplying the amount of thrust per motor by the number of motors (4) the maximum thrust of the quadcopter is able to be determined. In an ideal situation the weight of the quadcopter should be half of the maximum amount of thrust, meaning that at 50% throttle the quadcopter will be able to hover. By increasing the throttle, the quadcopter can be controlled to rise and by decreasing the throttle the quadcopter will start to fall.

The Sunnysky X2212 980KV Brushless motors are used on the Hovercam. Figure 2 below displays the motor information from the datasheet. These motors were selected because the total maximum output thrust is greater than 2kg, almost twice the estimated weight of the Hovercam. The calculations for thrust will be done using the specs for the 1047 propellers, the 9047 propellers don't provide as much thrust as the 1047, and with the selected frame the 1145 propellers will interfere with each other.

By increasing the voltage and current to the motors a maximum of 870g of thrust is achievable for each motor. This amount of thrust will be used as the 100% throttle state in the calculations. With 4 motors, the maximum thrust of the Hovercam is calculated below.

$$Maximum\ Thrust = (Thrust\ per\ motor) * (number\ of\ motors) = 870g * 4 = 3480g$$

In order for the Hovercam to have good reaction time the hover state needs to be able to be achieved at around 50% throttle. At 50% throttle, the available thrust is half of the maximum thrust, or 1740g. In order for the Hovercam to be able to hover at 50% thrust the weight of the quadcopter must be 1740 grams. The weights of all the components on the Hovercam are shown in Table 2 below. The total weight of the Hovercam comes to 1284.6 grams, which is much less than the 50% thrust of 1740g. Because the Hovercam weight less than 50% of the maximum thrust, the Hovercam will be able to hover using less current allowing the Hovercam to sustain flight for a longer period of time, which will be discussed later in the Battery section.

Table 2: System Weight

Parts	Weight Calculations	Weight (g)
Frame	282*1	282g
Motor	56*4	224g
Propeller	10*4	40g
ESC	27*4	108g
Flight Controller	38*1	38g
Navigation Controller	59.6*1	59.6g
Battery	490*1	490g
Camera	Included in Navigation Controller	N/A
Ultrasonic Sensors	8.5*4	34g
GPS Module	9*1	9g
Total		1284.6g

The motor and ESC datasheets call for 2-3 cells in series in order to operate at the recommended voltage. Each cell in a LiPo battery provides 3.7V, by having 3 of these cells in series 11.1V is able to be provided to the system which is needed to provide maximum thrust.

Each motor requires 13.2A to run at maximum thrust. With all 4 motors running the needed current is 52.8A. Because other parts of the battery will also require power an extra 5A is used in calculations, this estimate of a necessary 57.8A is well above what will be actually used.

Motor: X2212 KV:980

Technical Datas		Recommended Prop(inch)			
KV	980	Standard	2s-1145/1147 3s-1047/9047	Max thrust	
Configu-ration	12N14P				
Stator Diameter	22mm				
STator Length	12mm				
Shaft Diameter	3mm				
Motor Dimension(Dia.*Len)	Φ27.5×30mm				
Weight(g)	56g				
Idle current(10)@10v(A)	0.3				
No.of Cells(Lipo)	2-3S				
Max Continuous current(A)180S	15A				
Max Continuous Power(W)180S	150W				
Max. efficiency current	(7-12A)>78%				
internal resistance	126mΩ				

Tested with SunnySky motor 30A ESC

Prop	Volts (V)	Amps (A)	Watts (W)	Thrust (g)	Efficiency (g/W)			
9047	8.5	6.5	55	420	7.63			
	10	8.2	82	560	6.82			
	11.1	9.6	106.5	680	6.38			
	12	11.0	132	740	5.60			
1047	7.4	7.4	54.7	480	8.77			
	8	8.2	65.6	520	7.90			
	10	11.2	112	720	6.42			
	11.1	13.2	146.5	870	5.93			
1145	8.5	12.0	102	680	6.66			
	10	14.2	142	880	6.19			
	11.1	17.2	190.9	960	5.02			

Figure 2: Data table for the Sunnysky X2212 motors.

Electronic Speed Controllers (ESCs)

The ESCs used in the design are four HP SimonK 30A Speed Controllers. These ESCs have an extremely low output resistance and superior current endurance. Protection features of the ESCs include low-voltage cut-off protection, over-heat protection, and throttle signal loss. The low voltage cutoff will cut the power to the motors when the voltage drops to a specific level. This is a protection feature for LiPo batteries. If a LiPo battery's voltage drops below its minimal voltage, it can permanently damage the battery. This allows the motors to be protected if anything happens to the battery or ESC. Also the ESC provides a separate voltage regulator IC for the microprocessor providing a good anti-jamming capability. The throttle range can be configured to be compatible with all transmitters providing a smooth, linear, and precise throttle response. With a continuous allowed current of 30A and max current of 35A bursts the ESCs are rated well above the motor's current draw specified in Figure 2 above. The ESC calls for a 2S or 4S LiPo battery to operate the motors.

Propellers

On every quadcopter, there are two CW (clockwise) and two CCW (counter-clockwise) propellers. There are propellers of different length and pitch. The size of a propeller is measured from tip to tip, its diameter, as when you spin up a propeller you get a circle and the diameter is the size of the propeller. Pitch is sometimes called pitch length as well, which can be defined as the travel distance of one single propeller rotation. Generally, either increased propeller pitch or diameter will lead to higher current draw, because more air is moved and it gets harder to spin, assuming RPM is the same. In a nutshell, larger propeller or higher pitch length will increase your vehicle speed but also use more power. When deciding on propeller size, you need to find a good balance between pitch and length. Generally, a propeller with lower pitch numbers can spin faster (higher RPM), the motors don't need to work as hard to spin it so it pulls less current. If you want to do acrobatics, you will need lower pitch propellers which provide higher acceleration and it puts less pressure on the power system. Lower pitch propellers will also improve stability. A higher pitch propeller moves greater amount of air, which could create turbulence and cause more propeller wash. It generates more thrust in the expense of higher current draw, but giving you higher top speed. A smaller propeller is easier to stop and speed up while a larger propeller takes longer to change RPM due to inertia. The propeller chosen for the Hovercam is a 10" x 4.5 pitch propeller also referred to as a 1045. This propeller set is a High RPM version allowing for the motors and frame be better balanced. They are made of a strong Nylon-Carbon composite which is very a light weight material.

Using a propeller with a pitch of 4.5 was more beneficial for the motors in the kit for several reasons. Looking at thrust and efficiency there are advantages and disadvantages. Comparing the 1045 pitch to the 9047 in the motor spec sheet, shows that the 1045 has a higher thrust but lower efficiency. Comparing the 1045 with the 1145, the 1145 has a higher thrust but lower efficiency. As the main goal is to strive for thrust with respect to the weight of the components (3480g) the best choice would be to choose the 1145 propeller. With the 1145 propellers and the selected frame there would be some interference with the neighboring propellers as well as requiring more current, which would decrease the flight time of the Hovercam.

Battery

When selecting a battery there are other factors to take into account as well; the capacity and the charge/discharge rate. The capacity of a LiPo battery is measured in milliamp hours (mAh) and is the measure of how many milliamps the battery can provide for an hour. The charge rate is the rate that the battery can be either charged or discharged, in this application the discharge rate is most important. The discharge rating or C rating is used with the capacity in amp hours of the battery to determine the amount of amperes that the battery can safely discharge at once.

A 3S2P Lipo 8000mAh, 40C battery would be able to output a continuous current of 320A

$$Continuous\ Discharge\ Rate = (Capacity(Ah)) * (Constant\ C\ Rating) = 8.0Ah * 40 = 320A$$

This 320A is the maximum rate of current draw that the battery can safely supply. This is more than enough to operate the Hovercam that has a maximum current draw of 57.8A. The run time of the Hovercam is determined using the maximum current draw and the capacity of the battery.

$$Run\ time = \frac{Capacity\ (Ah)}{Maxximum\ current\ draw} = \frac{8.0Ah}{57.8A} = .138\ hours\ or\ 8.3min.$$

Each motor requires less than 7.4A to run at 50% thrust. With all 4 motors running the needed current is 29.6A. Because other parts of the battery will also require power an extra 5A is used in calculations, this gives a necessary 34.6A to run.

$$Run\ time = \frac{Capacity\ (Ah)}{Maxximum\ current\ draw} = \frac{8.0Ah}{34.6A} = .231\ hours\ or\ 13.87min.$$

The Hovercam will run using a combination of the two states, so the battery life will be somewhere between the hover state calculation and the maximum thrust calculation. The average of these flight times is 11.85min of flight. With a desired flight time of 10 minutes minimum, 1 of these batteries will be used giving the system about 11.85 minutes of flight.

A Floureon 3S2P 8000mAh 40C Lipo Battery will provide the Hovercam with 11.1V and will be able to operate the Hovercam for the timespan calculated above. After all the parts for the Hovercam were chosen the actual current draw of the system was calculated and is in fact less than the value used in the calculations above. Table 3 shows the needed current for the components of the Hovercam as well as the total current used by the system. Because this current is less than the value used in the calculations above for the maximum current draw, the Hovercam will be able to sustain flight slightly longer than previously calculated.

Table 3: System Current Draw

Parts	Current Draw	Total Current
Motor/ESC	13.2A*4	52.8A
Navigation Controller w/ Camera	2.5A	2.5A
Flight Controller	500mA	0.5A
Ultrasonic Sensors	2mA*4	.008A
GPS	20mA	0.02A
Total		55.83A

Flight Controller

The flight controller is responsible for controlling the motors so that the Hovercam keeps the tracked image in the scope of the camera and within the predetermined distance from the target. The flight controller has inputs from Ultrasonic sensors that are used as proximity sensors for use in obstacle avoidance. The flight controller also has inputs from a barometer so that the height of the Hovercam can be monitored and controlled. Inputs from a gyroscope and an accelerometer are used to monitor the roll, pitch, and yaw of the Hovercam. This data will be compared to the requirements set from the Navigation controller and will be used to control the motors. The yaw is used to control the direction that the Hovercam is facing. The yaw will be adjusted so that the Hovercam will have the target that is being tracked in the center of the field of vision of the camera. The roll and pitch are used to control the movement of the Hovercam. As the pitch of the Hovercam is adjusted the Hovercam will move either forward or backwards, positive pitch for forward movement and negative pitch for backwards movement. As roll of the Hovercam is adjusted the Hovercam will move either left or right, positive roll is for movement left and negative roll is for movement right.

The Pixhawk flight controller is selected for the Hovercam. The Pixhawk has an integrated gyroscope, accelerometer, barometer, and magnetometer. The Pixhawk flight controller will communicate with the Navigation controller through the TELEM2 port. The Pixhawk also have ports for up to 8 motors and 6 auxiliary outputs that can be used for the Ultrasonic sensors.

Smart Phone Application

Smart phone applications are used for all sorts of control purposes these days as smart phones are a commonality in everyday life. With free application development programs easily downloaded from the internet, creating a smart phone application to send basic commands to the quadcopter was definitely the most reasonable choice monetarily and when it comes to ease of use. No extra equipment being required would also be a relief to users making the product more desirable.

The smart phone application will be made utilizing Visual Studio and Xamarin. Visual Studio was chosen as the IDE to use because it has a lot of support for c# development, GUI design, and it has Xamarin support implemented. Xamarin is a framework that allows for native android development in c#. Xamarin was chosen because it will allow for a streamlined creation of a smartphone application without having to worry about the backend work of supporting a native android operating system.

The application GUI is structured quite simply for ease of use. The first screen prompts the user for a password so that the security protocol can be activated. The following screen will be the main controls screen featuring the Set Distance, Take Off, Land, and Hover buttons. Before anything else can be selected the user must set the distance between the smart phone and the quadcopter by clicking the button, being taken to a separate screen with a slider to set the distance, and then returning to the main controls screen. Now, the Take Off button is available. After being clicked, the quadcopter will then enter the stationary hover state. Landing is now an option in the state. Also located on the main controls screen are the buttons that allow the user to view the video feed and to select a target. The View Video Feed button merely takes the user to a different screen with a window displaying the video feed. The Target Selection button takes the user to a similar screen, but also allows them to select a target by pressing and dragging to form a box around the region of interest. After selecting a target, the quadcopter will enter the follow state where the Hover command becomes available and the Land command becomes unavailable. The Hover command sends a message to stop tracking and stay in position. This returns the quadcopter to the hover state and allows the user to click the Land button.

To ensure a secure connection between the smart phone and the quadcopter, Transport Layer Security (TLS) will be implemented. TLS is a cryptographic protocol that provides a private connection and maintains data integrity between two communicating devices. So that the quadcopter cannot be hacked into, TLS enacts a handshake protocol between the quadcopter and the smart phone at the start of each session that generates keys unique to each session. The encryption algorithm and cryptographic keys are determined by the two devices before any data is transmitted negating any potential hackers trying to take over during communications. Each of the devices can also be authenticated by utilizing public-key cryptography and the integrity of their messages can be maintained by implementing a message authentication code.

Navigation Controller

The Raspberry Pi 3 model B was chosen to be the processor for the navigation controller. The Raspberry Pi was chosen because it met the design requirements of using live imaging processing for fine target tracking. It includes a camera interface, microSD slot, 802.11b/g/n wireless LAN, 1 GB of RAM, and a 1.2GHz processor. It is also lightweight at only 2.1 ounces. The power requirements are minimal in that only five volts are required with a current draw from 1.2A to 2.5A. The 802.11 b/g/n wireless LAN card will allow for effective communication to the smartphone application, as the data rate can range from 11 to 300 Mbps depending on the receiving smartphone network adapter. The processing power of 1.2 GHz will allow the navigation controller to use multithreading to support all software features in the design. The 1 GB of RAM will allow for the image processing to utilize high resolution pictures because 1GB of RAM is plenty of space for the program to run along with image processing. The microSD slot will allow navigation controller to use a microSD card to record and store the video data.

The Raspberry Pi 8MP camera was chosen to use for the image processing method. The camera allows for the following video modes: 1080p30, 720p60 and 640 × 480p60/90. This will allow for high quality video and image processing if the navigation controller uses at least the 720p60 which is 720p with 60 frames per second. These different video modes also allow for the ability to send a lower resolution video feed to the smartphone application, which will in turn lower the power cost of transmitting data compared to streaming pure 1080p quality video. As the camera was created for the raspberry pi, there is open access with the camera API to read the video stream directly.

GPS is added to the navigation controller using the Adafruit Ultimate GPS hat. This GPS module was chosen because it is able to interface with the raspberry pi using UART to communicate effectively. This module is able to update at 10Hz, so the GPS data can be pooled ten times a second, which is fast enough for tracking a moving object. The current draw is only 20mA, so this barely affects the system's power draw from the battery.

D. Marketing Requirements

In order for the Hovercam to be a successful product, it needs to meet requirements that make it marketable. By meeting these requirements, the Hovercam becomes valuable to the customer, providing a service that had previously been left unfulfilled. Table 4 below displays the marketing requirements that the Hovercam will fulfill.

Table 4: Marketing Requirements for the Hovercam

Marketing Requirements	
1.	The device will film the target it is tracking.
2.	The device will follow a target within a user specified distance.
3.	The device will avoid obstacles in flight.
4.	The device will be lightweight and durable.
5.	The device will have a high quality video output.
6.	The device will have a rechargeable battery.

E. Objective Tree

The functionality of the Hovercam is broken down into several simple requirements. These requirements for the Hovercam are broken down into more simple and measurable objectives. These objectives are mapped into a flowchart in Figure 3 as the objective tree. By following the objectives in the objective tree the Hovercam will meet the marketing requirements mentioned above.

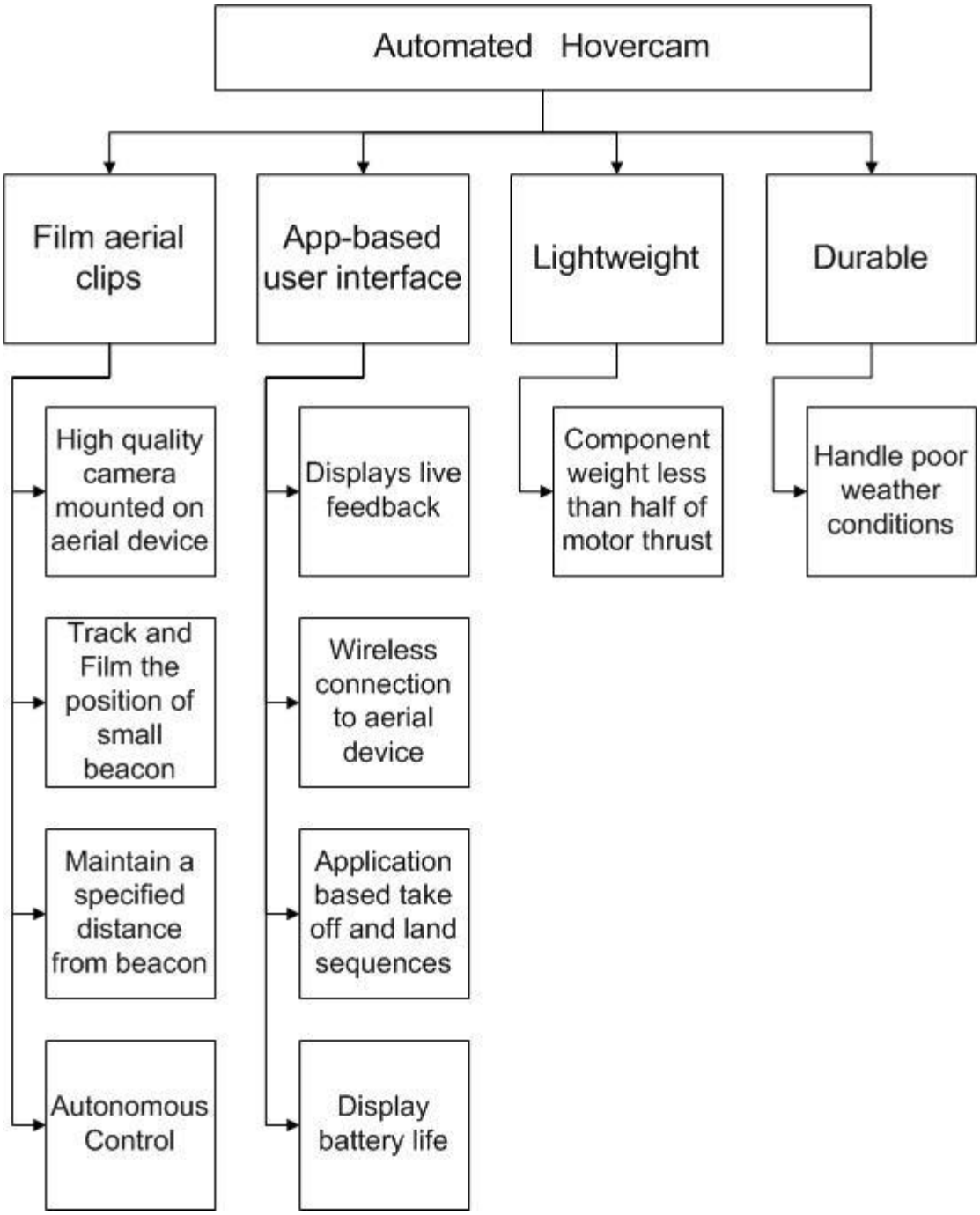


Figure 3: Objective Tree

6. Design Requirement Specifications:

The design requirements for the Hovercam listed below in Table 5 merge the marketing requirements with the technical designs that will fulfill those requirements. The engineering requirements are used to verify that each step of the technical design, in the following section, is pertinent to the operation and function of the Hovercam.

Table 5: Design Requirements

	Marketing Requirements	**Engineering Requirements**	**Justification**
1.	4, 5, 6	The battery must be able to sustain at least 10 minutes of flight.	The system must fly for at least ten minutes for actual use.
2.	2, 3	The device must be able to communicate with the smart phone within a range of 30m.	The system will typically be used outside and will need to communicate within the specified distance.
3.	1, 5	The device must be able to film the target using image processing while within a 10m range.	Video recording of the subject is crucial for both tracking and filming the object
4.	3, 4	The device must be able to fly outdoors without being affected by weather conditions.	The system must be able to fly in damp weather and windy conditions without breaking or being tossed around.
5.	2	The device must be able to take off, hover, and land with manual control.	The user will be able to tell the device to take off and safely land overriding the autonomous flight.
6.	1,2,5	The device must be able to reacquire the target using GPS if the target leaves the camera's field of view.	The objective is to film the target so, if visually lost, the target needs to return to the camera's field of view.
7.	5	The device must be able to stream video to the smartphone with a minimal delay of 100ms.	The video feed must not lag too far behind the movement of the device.
8.	2,5	The smartphone application must be executable on several different types of android smart phones.	The larger number of android phones this application works on the greater number of people will be able to use this device.
9.	1,5	The communication between the application and device must be secure	The system must be secure against others trying to crash or steal the

		against external interference.	device.
10.	2,3,4	The device must be able to detect and autonomously avoid obstacles within 2 meters of the device	The system must not crash into any stationary objects.

7. Technical Design

F. Hardware Level 0

The Hovercam hardware takes inputs of power control inputs and from sensors to fly autonomously. The Hovercam outputs thrust from the motors and video to the user. Figure 4 below displays the Level 0 block diagram of the Hovercam and Table 6 is the functionality table of the level 0 hardware diagram.

Figure 4: Level 0 Hardware Block Diagram

Table 6: Hardware Level 0, Automated Hovercam

Module	Automated Hovercam
Inputs	o Power o User Control Input o Sensors
Outputs	o Video Output o Motors o User Control Output
Functionality	This device calculates its position compared to the input signal position of the beacon, and it uses the motor controls to maintain a steady distance away from the beacon. User control input can be used to manually override the device, and the position and proximity sensors will stop the device from running into any obstacles. The video output will be sent as a live feed to the user control device, and the user control output will update the status on the user control device about the automated Hovercam.

G. Hardware Level 1

From the Hardware Level 0 diagram, the blocks have been broken down into manageable parts. The Level 1 flowchart below in Figure 5 depicts the direction of transfer of information and power in the Hovercam.

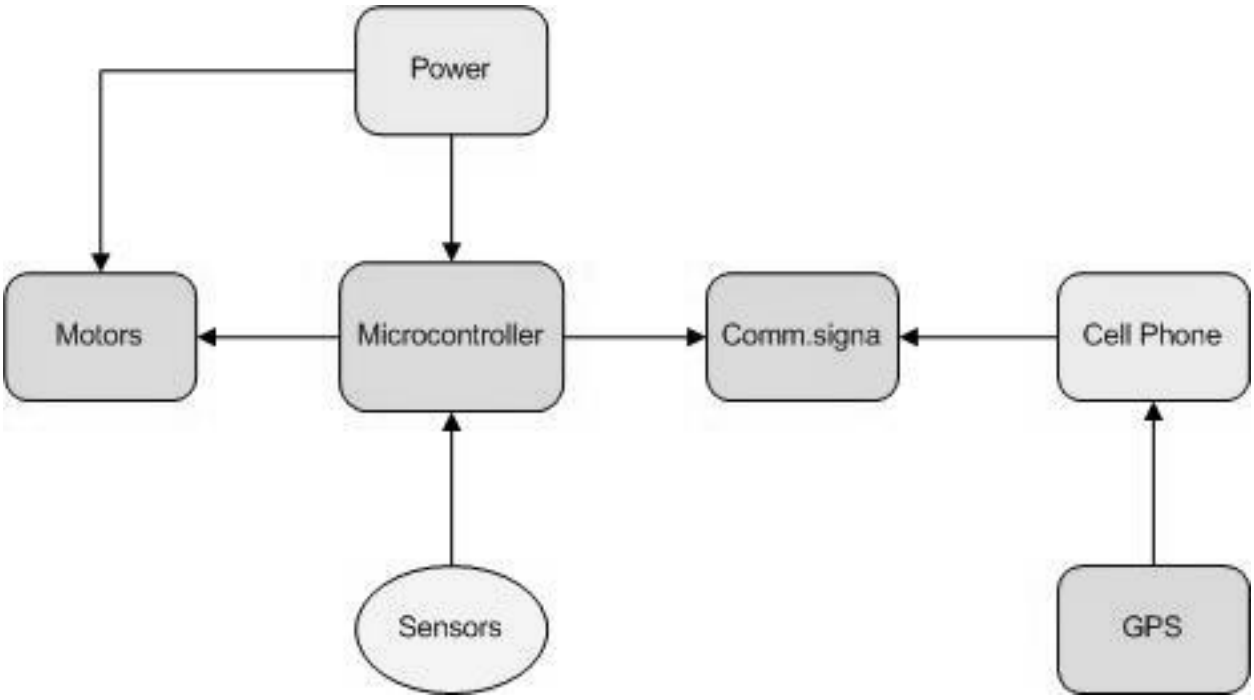


Figure 5: Level 1 Hardware Diagram

Each main block of the level 1 diagram above is broken down and explained in terms of its functionality, inputs, and outputs. Table 7 displays the microcontroller hardware and connections.

Table 7: Hardware Level 1, Microcontroller

Module	Microcontroller
Designer	Adam Fada
Inputs	Power (3.3-5V) Communications (Position and commands) Sensors
Outputs	Motor Communications (Video Signal)
Description	Microcontroller is collecting data from all of the sensors and determining the appropriate adjustments to the motors for flight tracking. The microcontroller is transmitting video from the camera to the cell phone controller.

Table 8 below depicts the necessary inputs and the outputs of the motors and how they operate in the system.

Table 8: Hardware Level 1, Motor

Module	Motor
Designer	Kevin Rauh
Inputs	Power Microcontroller
Outputs	Thrust to lift the quadcopter
Description	Electronic Speed Controls are getting power from the battery and commands from the microcontroller. The ESCs are then outputting power to the motors which the propellers are attached to. The ESCs are reading feedback speed data from the motors and feeding that back to the microcontroller.

Table 9 below depicts the functionality of the smart phone and how the smart phone will interact with the Hovercam while it is in flight.

Table 9: Hardware Level 1, Smart Phone

Module	Smart Phone Application
Designer	Ross Palenik
Inputs	GPS signal Video Feed from microcontroller
Outputs	Communications to Microcontroller
Description	Smart Phone allows user to choose a target to track and sends that information to the microcontroller on the quadcopter. It receives video feed from the quadcopter. Also allows user to send the take-off, land, and hover commands to the quadcopter microcontroller. Communication is protected using Transport Layer Security (TLS)

H. Hardware Level 2

The Level 1 Hardware diagram is broken down into the base parts needed for the Hovercam. Each piece in Figure 6 below shows the necessary connections for the Hovercam and flow of information from one piece to the next. The Tables below describe each of the main components in the Hovercam and how they relate to one another for the Hovercam to function properly.

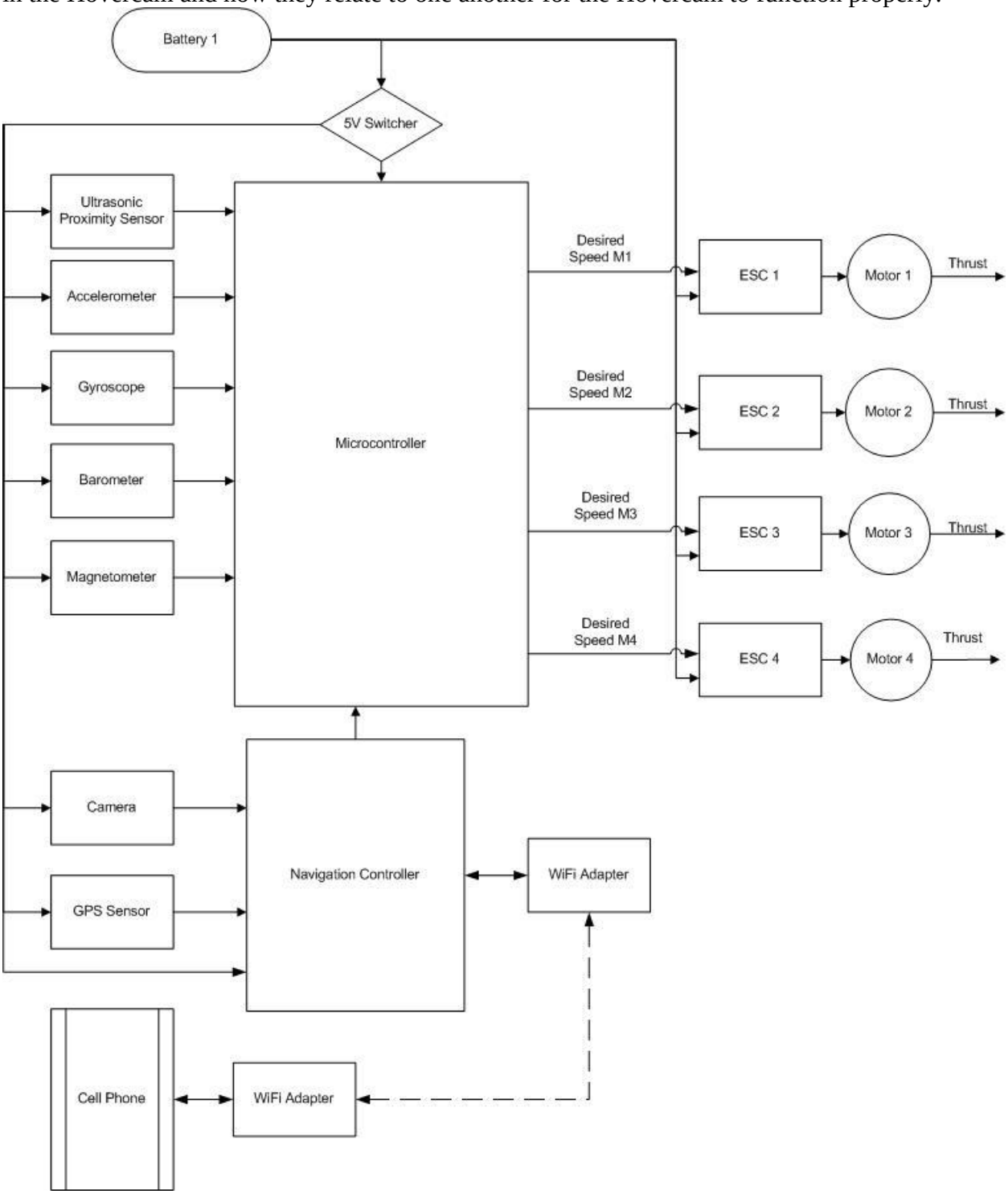


Figure 6: Hardware Level 2 Block Diagram

Table 10: Hardware Level 2, Electronic Speed Controller

Module	Electronic Speed Controller
Designer	Ross Palenik
Inputs	Power Microcontroller
Outputs	• Motors ○ Propellers
Description	The electronic speed controls are supplied power from the battery and commands from the microcontroller to provide a three phase low voltage source of energy to the motors and output specific controls to the motors. Attached to the motors are the propellers. The electronic speed controls will send data from the motors back to the microcontroller.

Table 11: Hardware Level 2, Navigation Controller

Module	Navigation Controller
Designer	Kevin Rauh
Inputs	Power- from Voltage Regulator Camera GPS Sensor Wi-Fi Adapter
Outputs	Microcontroller Wi-Fi Adapter
Description	The navigation controller is powered from the voltage regulator and is reading inputs from the Camera and GPS sensor to calculate the flight vector needed to maintain a predetermined distance from the user. The Navigation controller is then sending the flight control data to the Microcontroller and sending video feed to the smartphone through the Wi-Fi adapter.

Table 12: Hardware Level 2, Sensors

Module	Sensors
Designer	Kevin Rauh
Inputs	Power- from voltage regulator Gyroscope- measuring pitch, roll, and yaw Accelerometer- measuring the acceleration of the Hovercam Barometer- measuring the height of the Hovercam from the ground Ultrasonic Proximity Sensor- detecting if there are obstacles in the flight path Magnetometer- Uses Earth's magnetic fields for orientation
Outputs	Microcontroller
Description	All the sensors on the Hovercam are being powered from the voltage regulator. The sensors are acquiring flight information and sending the data to the microcontroller to determine corrective flight controls. The accelerometer is gathering speed and acceleration data, the gyroscope is measuring the pitch, roll, and yaw of the Hovercam. The Ultrasonic sensors are detecting any nearby obstacles, while the Barometer is measuring the height of the Hovercam.

Table 13: Hardware Level 2, Voltage Regulator

Module	Voltage Regulator
Designer	Kevin Rauh
Inputs	Power from battery
Outputs	Microcontroller Navigation Controller Ultrasonic Proximity Sensor Camera GPS sensor Accelerometer Gyroscope Barometer Magnetometer
Description	The voltage regulator is taking power from the LiPo battery and converting it 5V or 3.3V. The voltage regulator is then supplying that power to the microcontroller, navigation controller, ultrasonic sensor, camera, GPS, accelerometer, gyroscope, Magnetometer, and Barometer.

Table 14: Hardware Level 2, Motors

Module	Motors
Designer	Adam Fada
Inputs	ESC
Outputs	Thrust to Propellers
Description	The Brushless DC Motors (BLDC) are suppling a specific RPM limit with bi-directional current to the propellers for lift. The amount of thrust provided can change the Hovercam's height and orientation.

I. Software Level 0

The software of the Hovercam is used to control the movement and the video recording of the device. Figure 7 is the level 0 software diagram for the Hovercam, depicting the basic data inputs and outputs. Table 15 includes the basic description of the Hovercam software.

J.

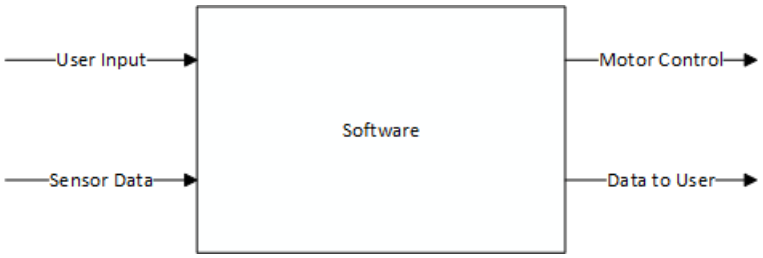


Figure 7 Level 0 Software Diagram

Table 15: Software Level 0, Hovercam

Module	Hovercam
Designer	Daniel O'Brien
Inputs	User Input Sensor Data
Outputs	Motor Control Signal Data to User
Description	The software takes in user input and sensor data to control the Hovercam. Also, data is sent to the user control device, including video and Hovercam status.

K. Software Level 1

From the level 0 diagram the software is broken down into 2 sub blocks. Figure 8 shows the main system blocks, one on the Hovercam itself and the other for the smart phone application. Table 16 contains information for the smartphone interface and how it communicates with the Hovercam. Table 17 below shows the Navigation controller on the Hovercam and Table 18 shows the Flight controller on the Hovercam.

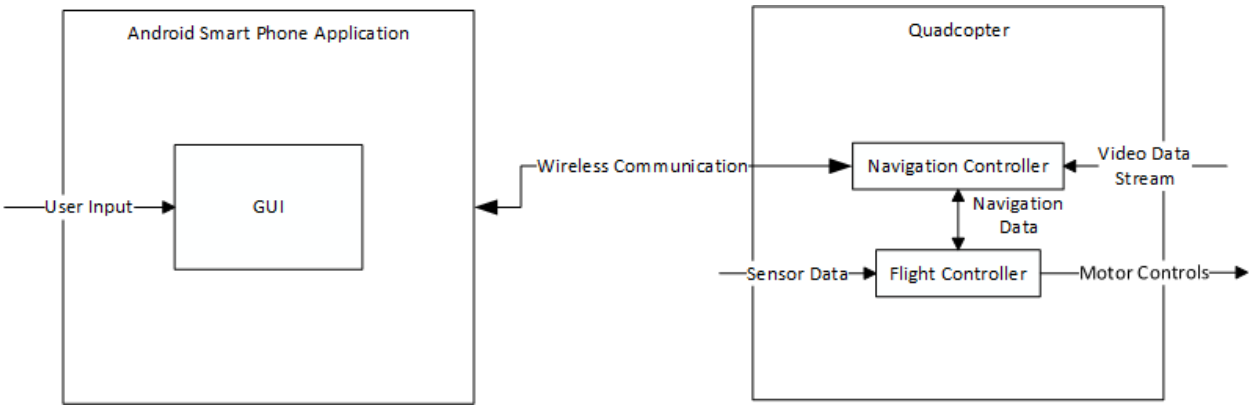


Figure 8: Level 1 Software Diagram

Table 16: Software Level 1, GUI

Module	GUI
Designer	Daniel O'Brien
Inputs	User Input Wireless Communication • Video Data • Location
Outputs	Wireless Communication • Commands • Location
Description	Takes in user input from the user to control the quadcopter. Displays video feed and location from quadcopter.

Table 17: Software Level 1, Navigation Controller

Module	Navigation Controller
Designer	Daniel O'Brien
Inputs	Video Data Stream Wireless Communication • Commands Location
Outputs	Wireless Communication • Video Data • Location Navigation Data
Description	Processes commands from android application to control the flight controller. Stores video data and streams it to the android application.

Table 18: Software Level 1, Flight Controller

Module	Flight Controller
Designer	Daniel O'Brien
Inputs	Sensor Data Navigation Data
Outputs	Motor Controls
Description	Processes sensor data to drive the motors using PWM control signals. Takes commands from the navigation controller.

L. Software Level 2

The following flowchart in Figure 9 describes the expected behavior of the smartphone application. The first step that the application must perform is creating a secure wireless connection to the quadcopter. The application will continuously wait for GPS coordinates requests from the quadcopter. The User Input Handler will consist of a GUI that will have commands the user can send to the quadcopter as described in Table 19.

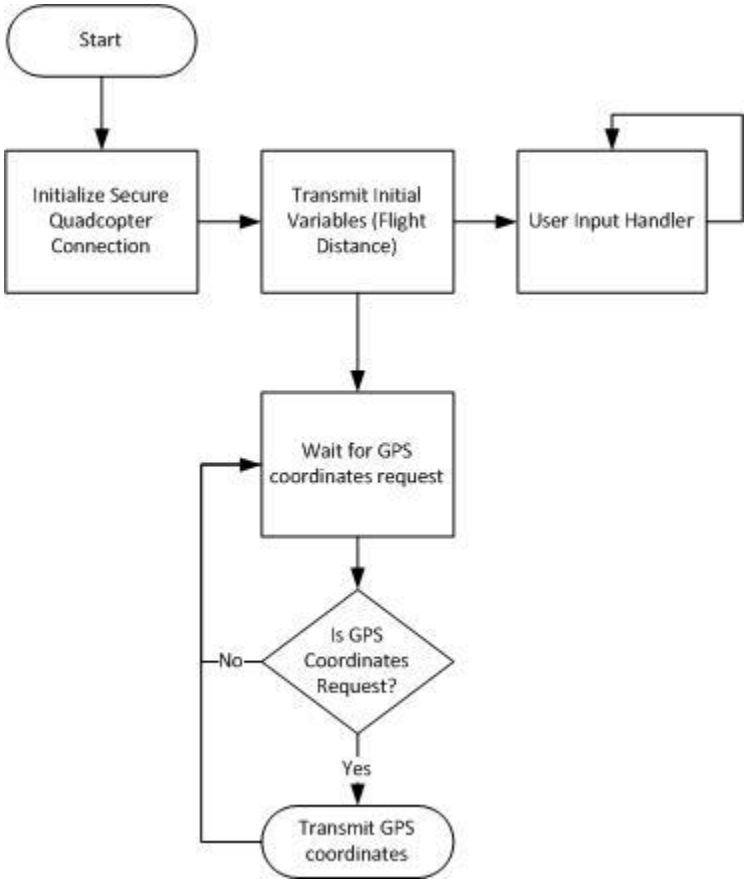


Figure 9: Smart Phone Application Level 2

Table 19: Smart Phone Application

Module	Smart Phone Application
Designer	Ross Palenik
Inputs	Wireless Communication • Video Data • Location
Outputs	Wireless Communication • Commands • Location
Description	The application will continuously wait for the GPS coordinates requests from the quadcopter. The user input handler will handle the sending of user commands to the quadcopter. The GUI for the application will feature buttons for each of those basic operation commands as well as screens to set the distance between the phone and the quadcopter, view the video feed, and select the target.

Figure 10 shows the flowchart of the navigation controller. The navigation controller starts and waits for a connection to be requested by the smartphone application. After the connection is initialized, a secure connection is created using TLS. Then, the initial variables, such as flight distance, are initialized. The navigation controller then creates four main threads that are ran in loops. The first thread waits for video feed requests and transmits it when requested from the smartphone application. The second thread handles user input from the smartphone application. The third thread continuously requests the GPS coordinates from the smartphone application and saves them in a variable for other threads to read. The final thread stores the video stream onto a microSD card, so the user can retrieve video footage later. The figure contains two subsystems, User Input Handler and Battery Life Check. Table 20 below displays the main functions used be the Navigation controller.

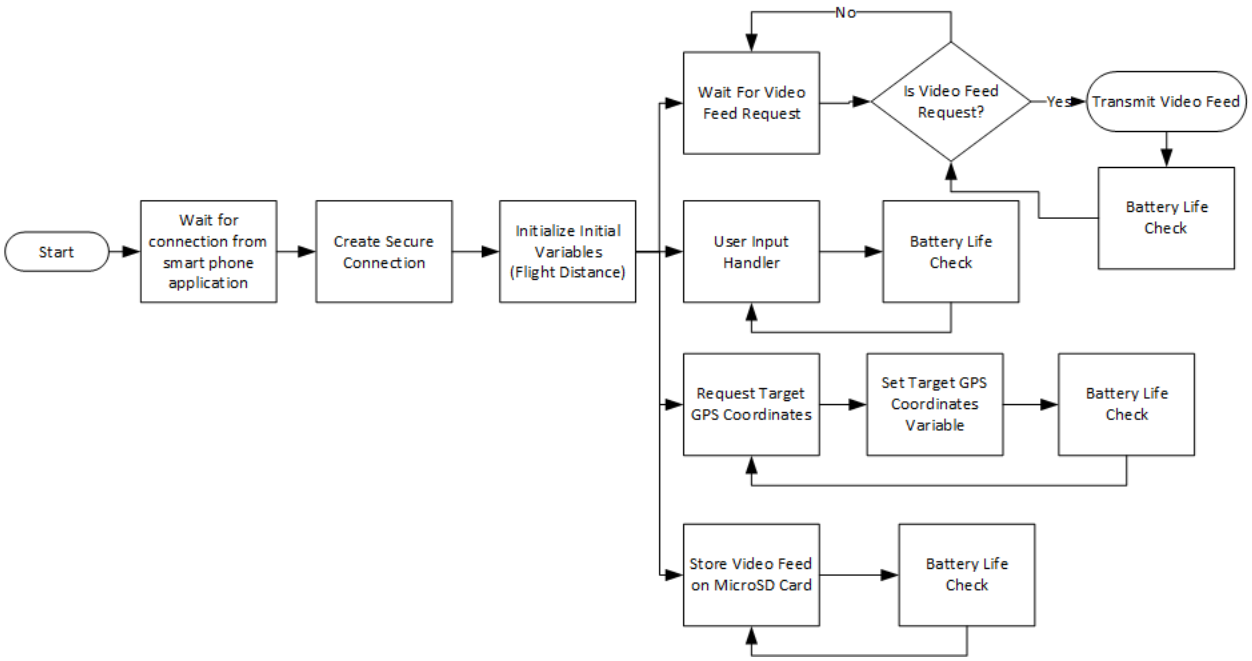


Figure 10: Navigation Controller Main Flowchart

Table 20: Navigation Controller functions

Function Name	Parameter	Description
WaitForConnectionFrom SmartphoneApplication	None	This function waits for a connection from the smartphone application. When it is created, the CreateSecureConnection function is called.
CreateSecureConnection	Socket Information	This function takes the created socket information for a TCP connection and creates a secure connection using TLS.
InitializeInitialVariables	Flight distance	This function takes in the flight distance and initializes it as well as other default variables used such as target GPS coordinates, quadcopter GPS coordinates, height, and battery life.
WaitForVideoFeed Requests	Socket Information	This function creates another socket with the smartphone application and waits for video feed requests from the smartphone application. When requests are received, it transmits the video stream.
UserInputHandler	Socket Information	This function maintains the current socket information or connection and handles any commands that are sent to the navigation controller from the smartphone application.
RequestTargetGPS Coordinates	Socket Information	This function creates another socket with the smartphone application and continuously requests GPS coordinates. The GPS coordinates are then saved in a variable for future use from other threads.
StoreVideoFeedOnMicro SDCard	Video Stream	This function writes the incoming video stream to memory on the MicroSD card.

The following flowchart in Figure 11 describes the flight controller that is on the quadcopter with the navigation controller. The fight controller handles the basic flight movement of the quadcopter based on the flight vectors that are sent from the flight controller. The flight vectors will be used to calculate the speed of each motor, so that the quadcopter moves in the correct direction. Also, it will use its sensors to avoid obstacles and maintain a stable flight.

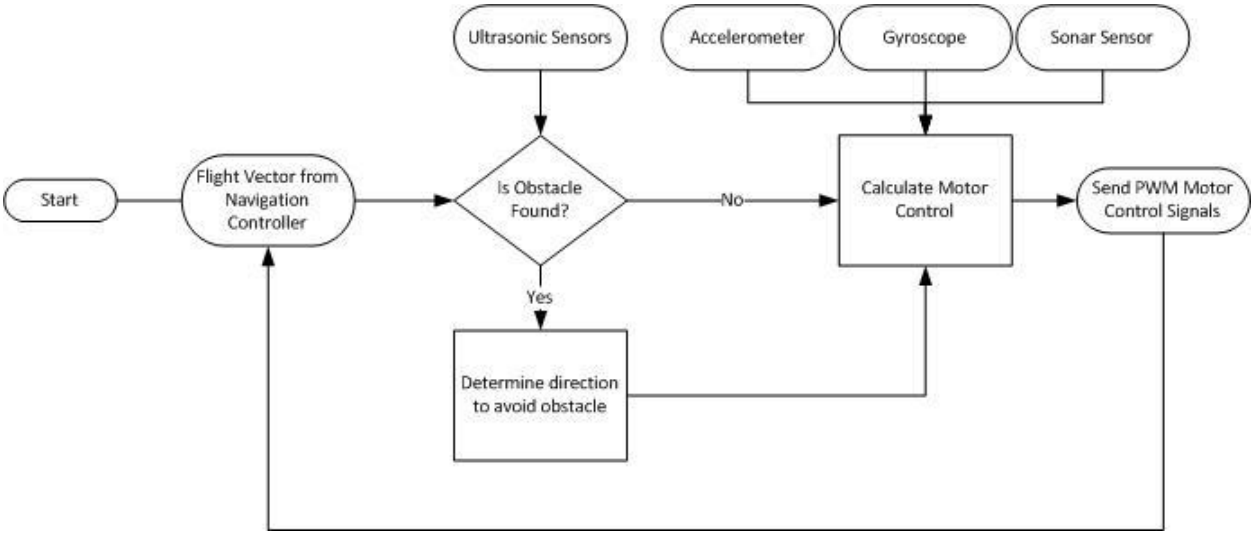


Figure 11: Software Level 2 Flowchart, Flight Controller

M. Software Level 3

The smart phone application is used to send commands to the navigation controller. The first thing the application does on startup is initialize a secure connection between the quadcopter and itself. This will require a password from the user. After transmitting initial variables, the quadcopter will continuously wait for the quadcopter to request it's GPS coordinates and transmit them when requested. The user interface will now be available as well allowing the user to send commands to the quadcopter via the user input part of the flowchart. The user input handler in the smart phone application is broken down into more detailed steps for control of the Hovercam. Figure 12 shows the smart phone application with the user input handler subsystem. This subsystem handles the user inputs that are to be sent to the quadcopter. The Take Off, Hover, and Land inputs each are buttons in the application and each send their respective commands to the quadcopter navigation controller to decipher. The Video Feed button takes the user to a separate screen where the application continuously requests the video feed from the quadcopter until the user leaves that screen. The Set Distance button takes the user to a screen where a slider can be used to set the desired distance between the smart phone and the quadcopter and that information is sent to the navigation controller. The Select Region of Interest button will take the user to a separate screen displaying the video feed. The user will be able to click and drag to form a square around the desired target. This information is then sent to the navigation controller. Table 21 below displays the functions used by the Smart Phone Application.

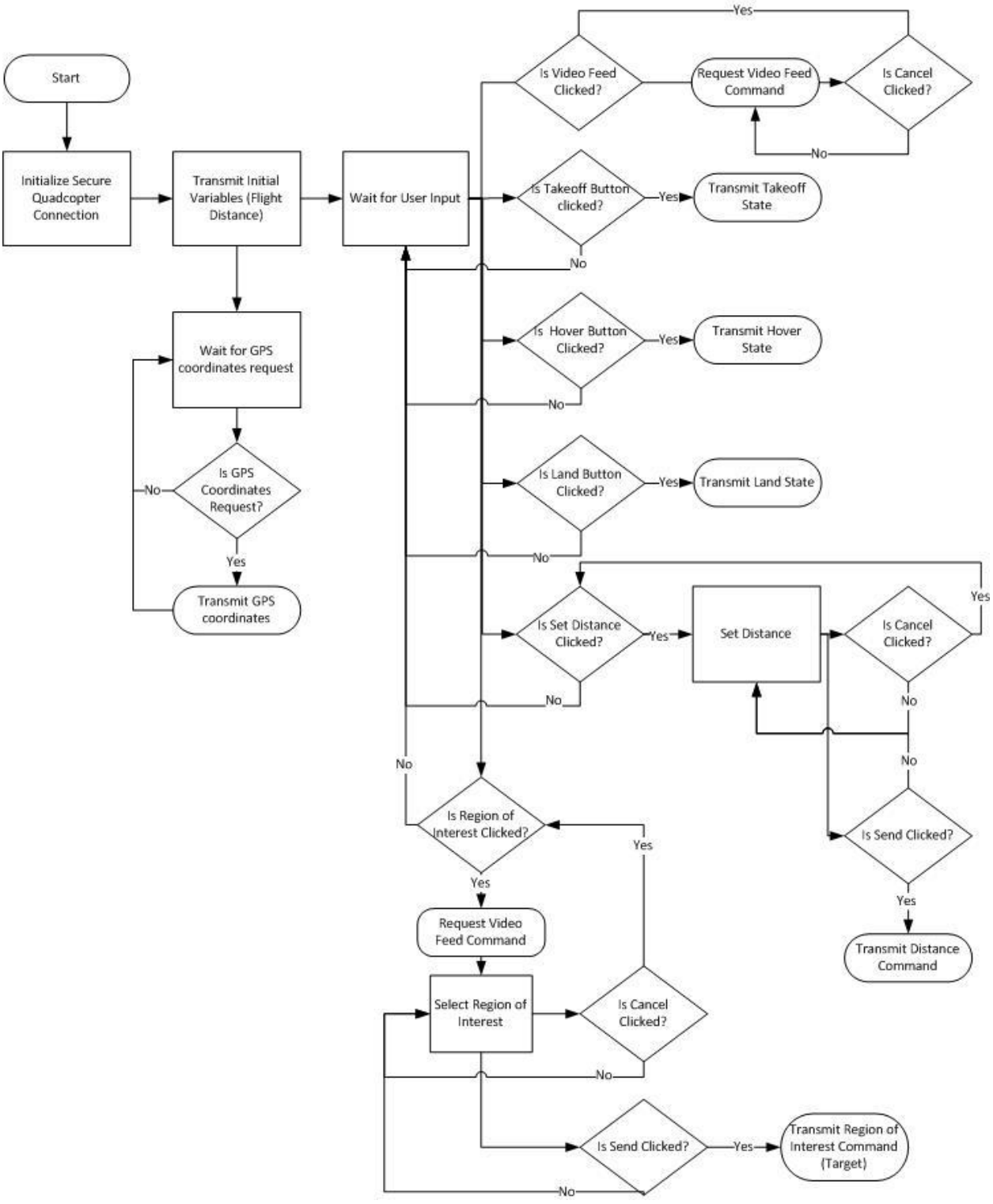


Figure 12: Smart Phone Application Flowchart

Table 21: Smart Phone Application Functions

Function Name	Parameters	Description
IsTakeOffClicked	Command Byte Stream	This function sends takeoff state opcodes to navigation controller.
IsHoverClicked	Command Byte Stream	This function sends hover state opcodes to navigation controller.
IsLandClicked	Command Byte Stream	This function sends land state opcodes to navigation controller.
IsSetDistanceClicked	Command Byte Stream	This function sends set distance state opcodes to navigation controller.
IsRegionOfInterestClicked	Command Byte Stream	This function sends region of interest state opcodes to navigation controller.
RequestVideoFeed	Video Byte Stream	This function requests video feed data from the quadcopter microcontroller.

The navigation controller will the system that communicates with the smartphone application over Wi-Fi and interfaces with the flight controller on the quadcopter. Wi-Fi was chosen because smartphones have access to a WLAN network card, and will allow for communication over long distances outside. The navigation controller sends flight commands to the flight controller to control the movement of the quadcopter. This process will be done using a combination of GPS navigation and image processing. GPS has an accuracy of around 7.8 meters at a 95% confidence, and that is not good enough to accurately record and track an object. To solve this, image processing will be used for the fine tracking of an object.

The image processing will be implemented using the OpenCV library, which stands for open source computer vision. This was chosen as it is compatible with the chosen Raspberry Pi processor and meets the design requirements of live image processing. The OpenCV library contains many object tracking methods, however, it was determined that the TLD tracking method fit the project parameters best. This is because the TLD tracking method is useful for tracking an unknown object in an unconstrained video stream. This accurately describes the tracking required for this project, as the object to be tracked will be dynamically selected within an unknown space of movement. TLD, tracking-learning-detecting, is a process in which an object is learned, tracked, and detected long-term. Using a medium flow algorithm, a region of interest will be selected as a bounding box in which to learn an object. This object will then be tracked from frame to frame, while the detector localizes all appearances of the object to identify any errors and learn. This will allow for improved tracking to prevent loss of object detection in future frames.

The navigation controller will be a multithreaded program running on a Linux distribution operating system. Using a Linux based distribution for the operating system will allow for less overhead of running the program while still maintaining the functionality to do computation, communicate with the smart phone application, and communicate with the flight controller.

Figure 13 shows the battery life check subsystem that is run on each thread for safety. This subsystem will check if the battery life is below a certain threshold, such as 10%, and will perform an emergency landing if it is. An emergency landing will stop powering all non-vital systems and land immediately. Table 22 below displays the functions used for the battery life check.

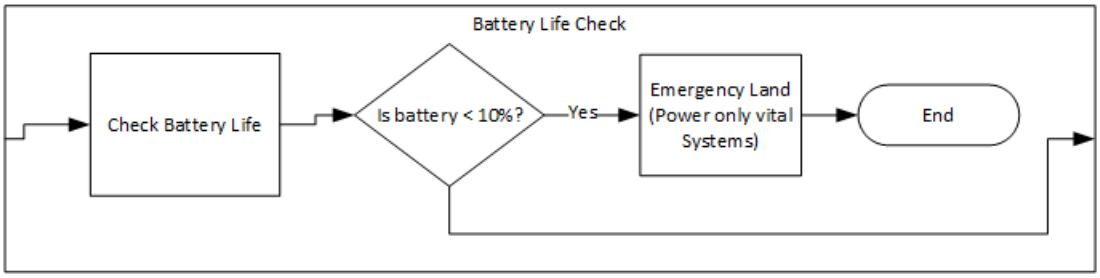


Figure 13: Battery Life Check Subsystem:

Table 22: Battery Life Check Functions

Function Name	Parameters	Description
CheckBatteryLife	None	This function will get the current battery life of the system.
IsBatterLifeLessThanThreshold	Current battery life	This function will determine if the battery life has passed the safety threshold, and it will return a Boolean.
EmergencyLand	None	This function will turn off power to all non-vital systems and perform a landing.

Figure 14 shows the user input handler subsystem. This subsystem handles commands sent from the smartphone application. The commands handled are takeoff, land, hover, set distance, and set region of interest. Each command will have its own opcodes to differentiate between commands. This subsystem also contains another subsystem block, flight controller handler. Table 23 below displays the functions used by the User Input Handler.

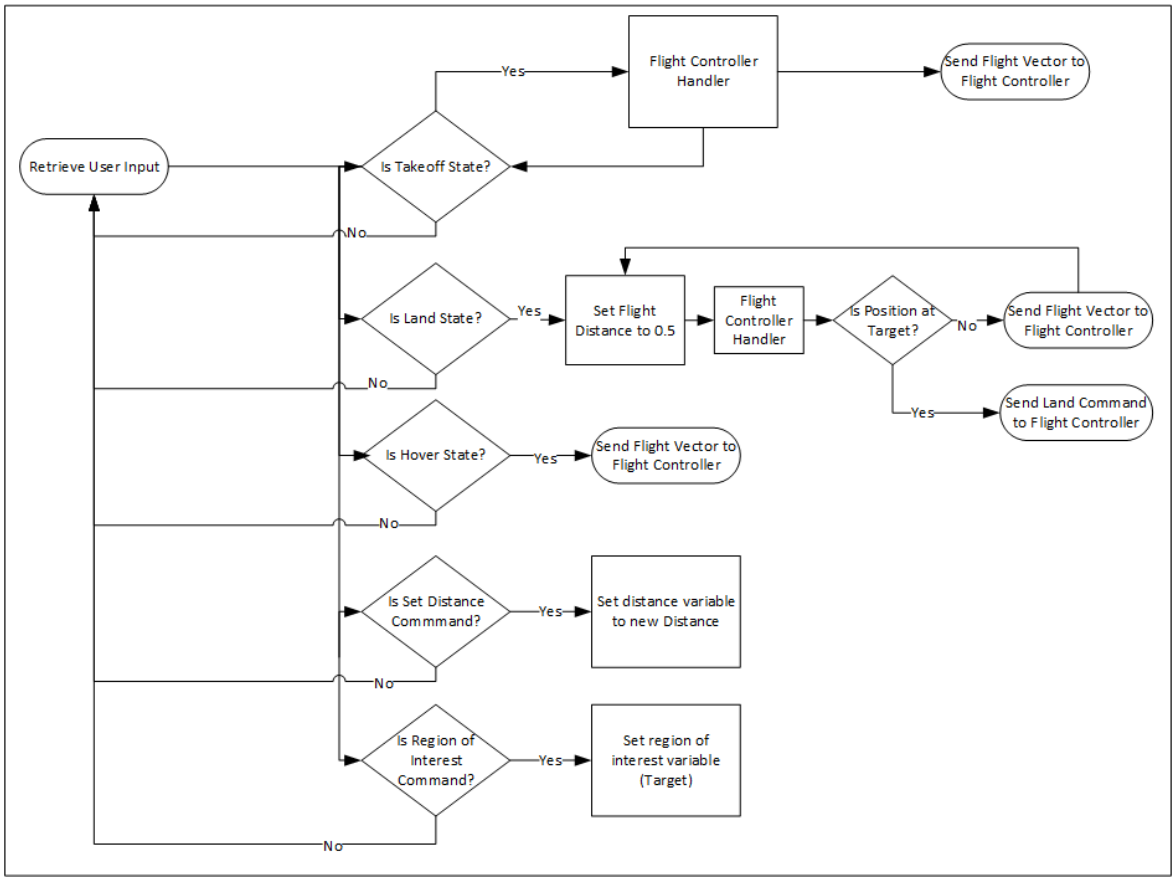


Figure 14: User Input Handler Subsystem

Table 23: User Input Handler Functions

Function Name	Parameters	Description
RetrieveUserInput	Byte stream	This function will handle the incoming byte stream from the application and send it to all of the command state handlers.
IsTakeOffState	Command Byte Stream	This function determines if the bytes sent contain the takeoff state opcodes. It returns a Boolean.
IsTakeOffState	Command Byte Stream	This function determines if the bytes sent contain the takeoff state opcodes. It returns a Boolean.
IsLandSate	Command Byte Stream	This function determines if the bytes sent contain the land state opcodes. It returns a Boolean.
IsSetDistanceCommand	Command Byte Stream	This function determines if the bytes sent contain the set distance state opcodes. It returns a Boolean.
IsRegionOfInterestCommand	Command Byte Stream	This function determines if the bytes sent contain the region of interest state opcodes. It returns a Boolean.
SetFlightDistance	Float	This function sets the distance variable
IsPositionAtTarget	None	Determines if the quadcopter is near the target (within 7.8 meters accuracy). It returns a Boolean.
SetRegionOfInterest	Rectangle	This function sets the bounding box of the region of interest for the TLD tracker.

Figure 15 shows the flight controller subsystem. This subsystem handles the image processing and flight vector calculation for controller the flight controller. The image processing algorithm determines if the tracked object is to the left or right side of the origin of the camera, with a 5% region of tolerance for the center. If the object is not in the center region, the flight vector will be influenced to change the yaw to orient either left or right. If the object is no longer detected, the flight vector fusion will use just the GPS data to track the target. The GPS coordinates are used to calculate the distance between the quadcopter and the target on the horizontal plane. If the target is within the specified distance, the quadcopter will move back, while if it exceeds the specified distance, the quadcopter will move forwards. There is, of course, a default region around the specified distance in which the quadcopter may hover without following the target if the target is not in motion. Table 24 below displays the functions used by the Flight Controller Handler.

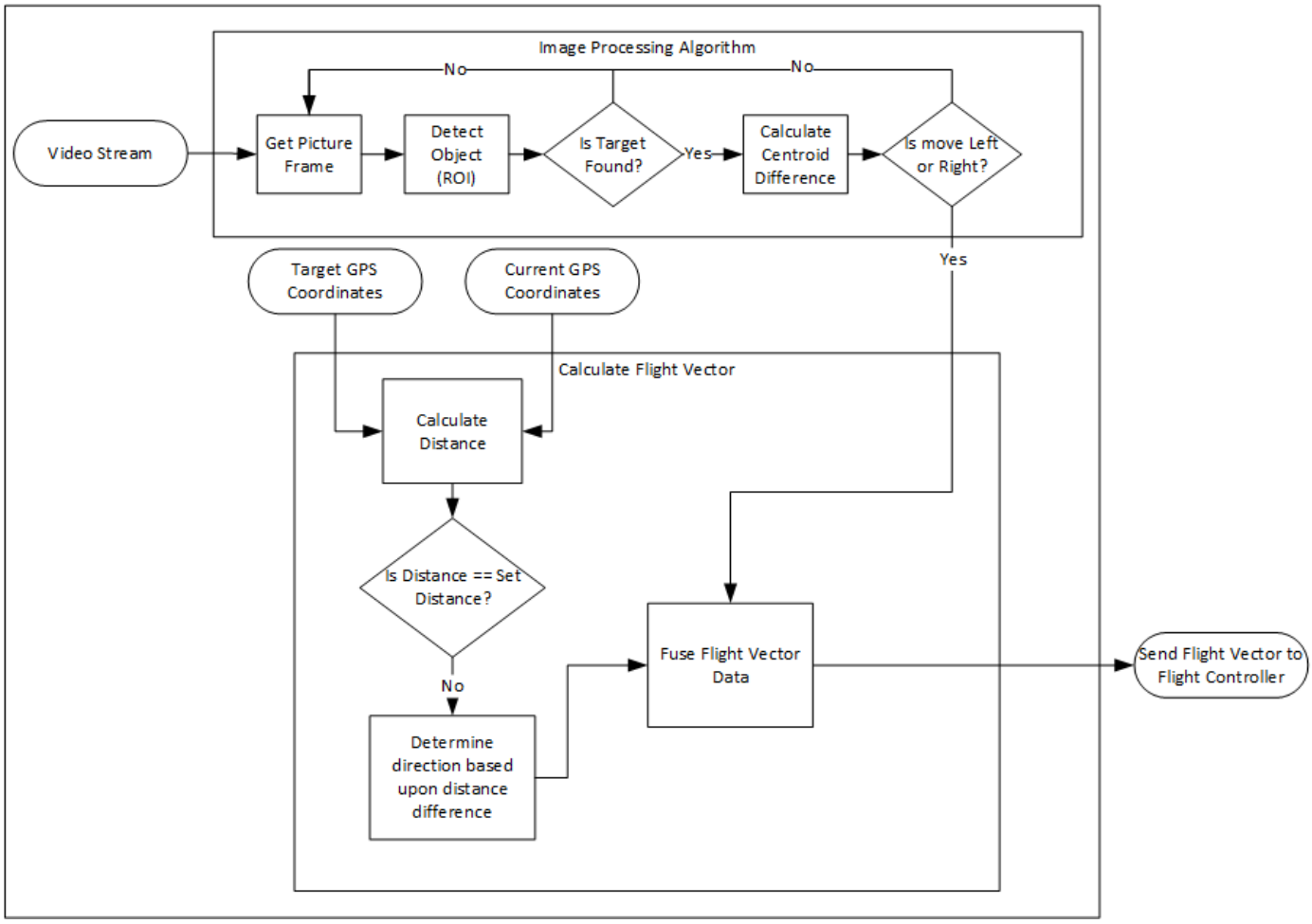


Figure 15: Flight Controller Handler

Table 24: Flight Controller Handler Functions

Function Name	Parameters	Description
GetPictureFrame	Video byte stream	This function grabs a frame from the video stream.
DetectObject	Video Frame	This function detects the object's position in the video frame. It returns x and y coordinates.
IsTargetFound	Object position	This function determines if the object was found within the video frame. It returns a Boolean.
CalculateCentroidDifference	Object position	This function calculates the position of the tracked object in reference to the origin of the camera. It returns the difference.
IsMoveLeftOrRight	Position difference	This function determines if the yaw should orient left or right based upon the centroid difference return value. It returns a Boolean.
CalculateDistance	Target GPS coordinates, quadcopter GPS coordinates	This function determines the difference of the horizontal plane between the target and quadcopter GPS coordinates. This returns the difference as a float.
IsDistanceEqualToSetDistance	Distance	This function determines if the distanced found from calculate distance is equal to the set distance, with a percent of margin error.
DetermineDirectionBased UponDistanceDifference	Distance difference from GPS	This function determines the direction, forwards or backwards, that the quadcopter must move to maintain a specified distance away from the target.
FuseFlightVectorData	Direction, Orientation	This function takes the direction provided by the GPS difference and the orientation from the image processing to determine the flight vector the flight controller should move.
SendFlightVectorToFlight Controller	Flight vector	This function sends the flight commands to the flight controller to move the quadcopter.

The flight controller monitors the current position and orientation of the Hovercam and compares it to inputs from the navigation controller and the proximity sensors. These compared values are used to control the thrust output of each motor for different states of flight. If an object comes within 2 meters from the Hovercam, detected by the ultrasonic sensors the Hovercam will enter an avoidance state. To avoid the object, the roll of the Hovercam will either increase or decrease depending on which sensor the object is detected on. If the object is detected on either of the right sensors the Hovercam will gain positive roll to fly left of the object. If the object is detected on either of the left sensors the Hovercam will gain negative roll to fly to the right of the object. In either avoidance direction, the pitch of the Hovercam will be adjusted to half of the set value. Once the ultrasonic sensors detect that the object is no longer present, the roll and pitch will return back to the value being sent from the navigation controller. The barometer on the Pixhawk will be used to monitor the height of the Hovercam. The height the Hovercam will be set to maintain is 5 meters. If the height is measured to be too low the thrust of the motors will increase until the correct height is achieved and if the height is measured to be too high the thrust of the motors will decrease until the correct height is achieved. After the correct height is achieved then the thrust of the motors will return back to their original state. Figure 16 below depicts the flow of operation for the flight controller.

Table 25 below displays the functions used by the flight controller.

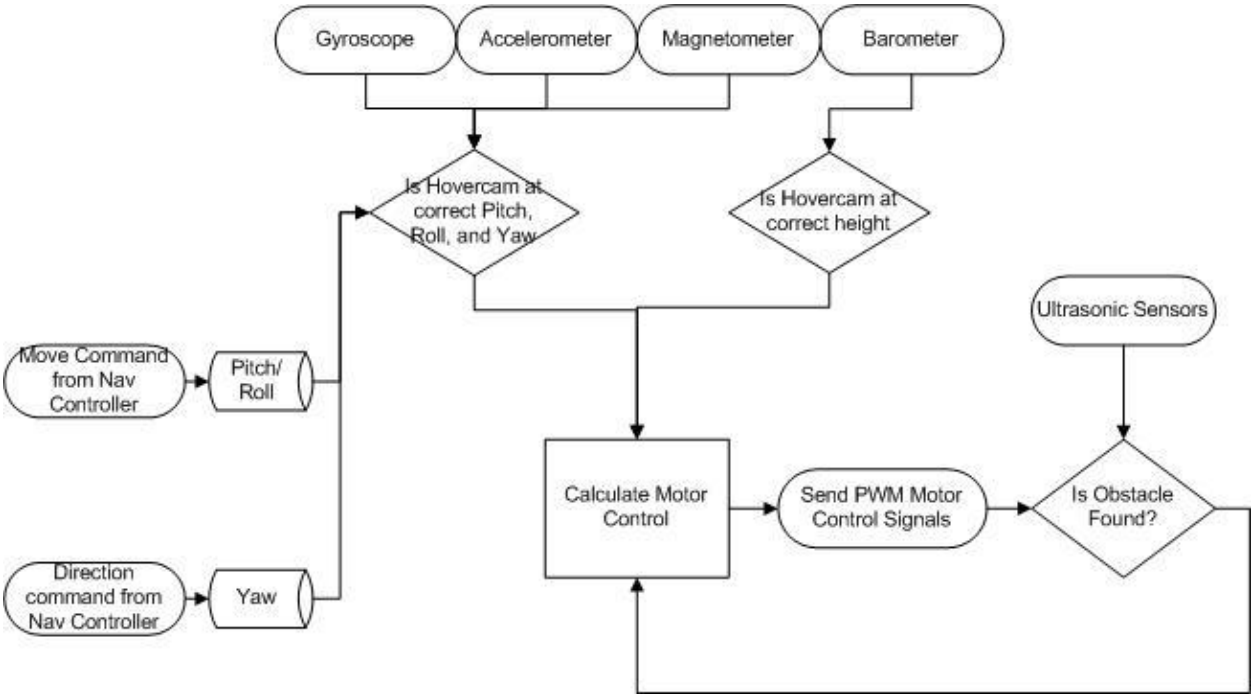


Figure 16: Level 3 Flight Controller Flowchart

Table 25: Flight Controller Functions

Function Name	Parameters	Description
IsObstacleFoundOnLeft	Left side Ultrasonic Sensor above threshold	This function will calculate if there is an object to the left of the Hovercam. Will enter the AvoidLeft function
IsObstacleFoundOnRight	Right side Ultrasonic Sensor above threshold	This function will calculate if there is an object to the right of the Hovercam. Will enter the AvoidRight function
AvoidLeft	IsObstacleFoundOnLeft	If the object is detected on the left sensors the Hovercam will gain negative roll to fly to the right of the object
AvoidRight	IsObstacleFoundOnRight	If the object is detected on the right sensors the Hovercam will gain positive roll to fly to the left of the object
AtHeight	Barometer	If Barometer reads below 5 meters enter GainAltitude function, if Barometer reads above 5 meters enter DecreaseAltitude function
GainAltitude	Below 5 meters	Increase all 4 motors output thrust by 10% of maximum thrust
DecreaseAltitude	Above 5 meters	Decrease all 4 motors output thrust by 10% maximum thrust
CheckPitch	Pitch from Gyroscope, Pitch from Navigation controller	Verify the pitch of the Hovercam is at the level sent from the Navigation controller. If they are not the same the pitch is adjusted
CheckRoll	Roll from Gyroscope, Roll from Navigation controller	Verify the roll of the Hovercam is at the level sent from the Navigation controller. If they are not the same the roll is adjusted
CheckYaw	Yaw from Gyroscope, Yaw from Navigation controller	Verify the yaw of the Hovercam is at the level sent from the Navigation controller. If they are not the same the yaw is adjusted

8. Accepted Technical Design

Hardware Design

Voltage Regulator [Kevin Rauh]

The Voltage regulator is responsible for regulating the battery voltage down from about 12V to 5V. A LM2576 step down voltage regulator is used, the switching regulator outputs 5V at 3A maximum load and can have a input voltage range of anywhere between 7V and 40V . The maximum output voltage from the battery is 12.6V which is well below the maximum for the regulator and a fully discharged 3S2P LiPo battery is 9.00V, above the minimum 7V that is necessary for the regulator. Figure *17* below shows the layout of the Voltage regulator circuit.

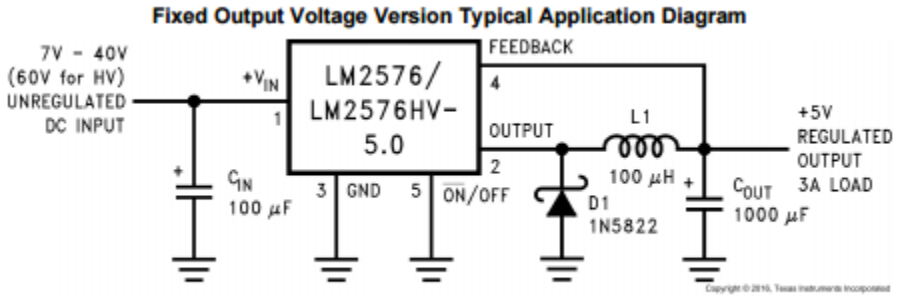


Figure 17: Voltage Regulator Circuit

The input voltage, pin 1 on the regulator will be connected to the battery and the regulated output voltage, pin 4, will be connected to both the navigation controller and the flight controller. The physical circuit for the voltage regulator is shown below in Figure 18.

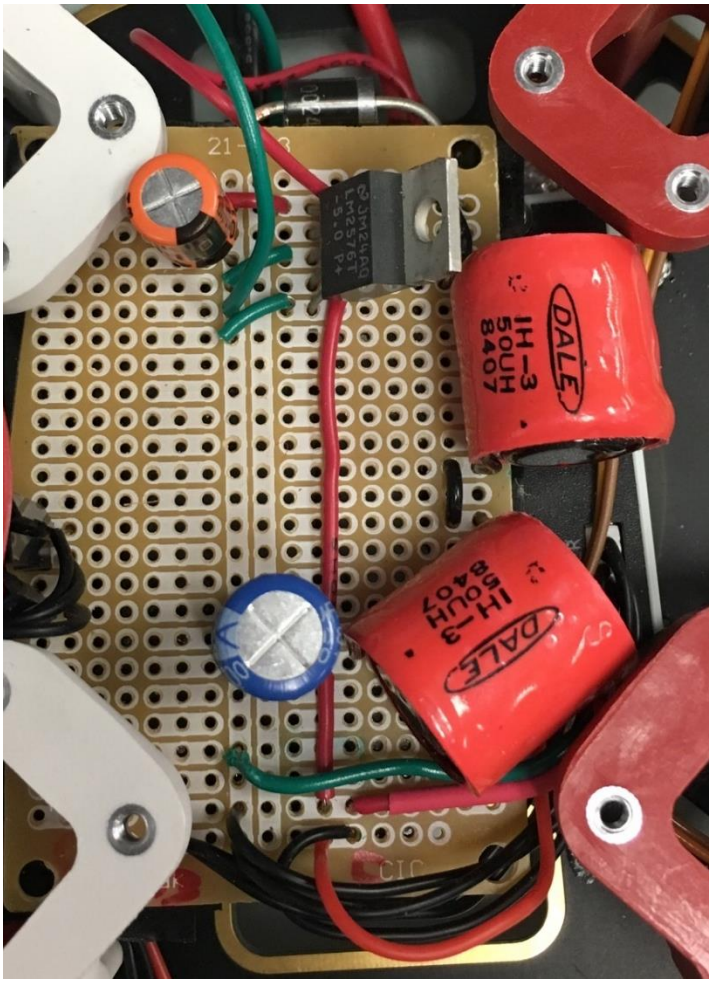


Figure 18: 5V switcher circuit

ESC Wiring [Kevin Rauh]

For a Quadcopter to fly, some motors must spin clockwise and some counterclockwise. The motors spinning in opposite directions keeps the quadcopter stable in flight and will keep it from rotating. To achieve proper stability motors opposite each other must spin the same direction. The Hovercam's motors are spinning such that the front right and back left motors are spinning counterclockwise, and the top left and bottom right motors are spinning clockwise. For the motors to spin in the proper direction, they must be wired to the ESC's in the proper fashion. Figure 19 below depicts the proper wiring for each rotational direction of the motors.

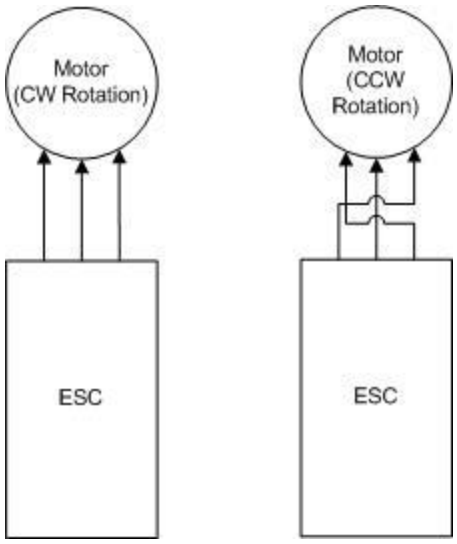


Figure 19: Wiring of the ESC's

Final Schematic of Hovercam [Adam Fada]

The complete circuit layout for the Hovercam is shown below in Figure 20. The voltage source in this circuit is a 3S2P LiPo battery that is directly powering the ESCs, and voltage regulator circuit.

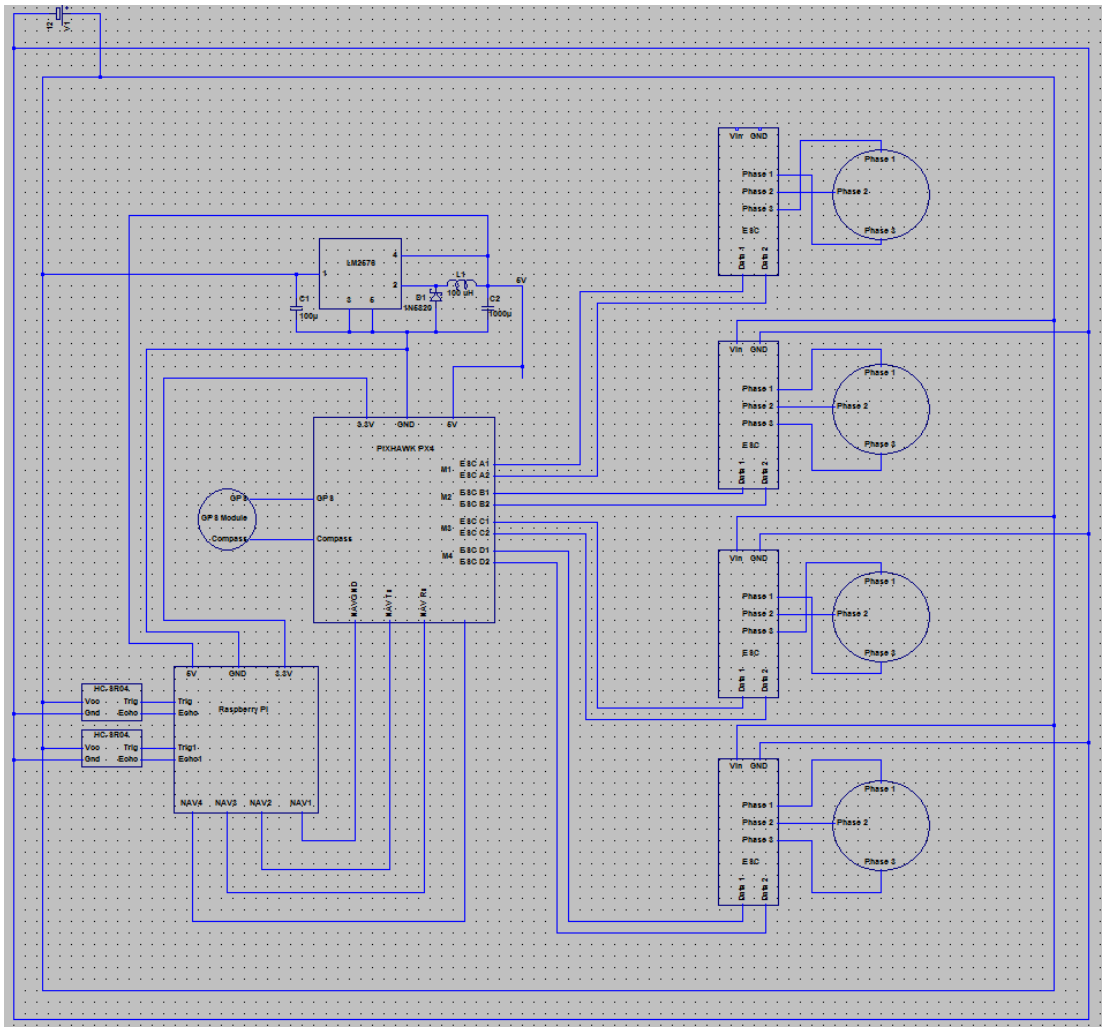


Figure 20: Schematic for Hovercam

Required Parts [Kevin Rauh]

In the preliminary design of the Hovercam, Table 26 was constructed listing the parts that were necessary for the construction of the project. As the design of the Hovercam progressed new parts were needed to fill design gaps in the original plan. Table 27 below is the final parts list for he Hovercam.

Table 26: Estimated Parts List

Qty.	Part Num.	Description
1		DJI 450 Frame Kit multi-rotor Wheel ARF kit
1		Pixhawk PX4 32Bit ARM Flight Controller
1	114990584	Raspberry Pi 3
6	SEN-13959	Ultrasonic Sensor 40kHz
1	2324	GPS HAT for Raspberry Pi
1		Floureon 2 packs 3s2p 8000mAh 40C Lipo Battery

1		Raspberry Pi Camera V2 Video module

Table 27: Final Parts List

Qty	Part Number	Description
1		DJI 450 Frame Kit multi-rotor Wheel ARF kit
1		Pixhawk PX4 32Bit ARM Flight Controller
1	114990584	Raspberry Pi 3
6	SEN-13959	Ultrasonic Sensor 40kHz
1		Floureon 2 packs 3s2p 8000mAh 40C Lipo Battery
1		Raspberry Pi Camera V2 Video module
1		Male to Female Jumper Wire 40pcs
1		GPS and Compass for Pixhawk
1		Male Deans Connector 2 pack
1		Micro USB cable
1		9450 Propellers 4 pairs
4		3D printed Ultrasonic Sensor mounts
1	LM2576	Series SIMPLE SWITCHER 3-A Step-Down Voltage Regulator
2		50 uH Inductor
1		100uF Capacitor
1		1000uF Capacitor
1	1N5822	Schottky Barrier Plastic Rectifier

Added to the original parts list were items needed to construct a Voltage regulator as discussed previously. Spare propellers were also needed because in testing of the Hovercam some of the original propellers were broken. The original GPS hat for the Raspberry Pi did not fit on our Hovercam so a different style was used of the final product. To wire the ultrasonic sensors and the Pixhawk to the Raspberry Pi, female connectors were needed because the GPIO pins on the Raspberry Pi had male leads.

Software Design

The software used in the accepted technical design of this project consists of a smartphone application, navigation controller, and flight controller. These three components are used to autonomously control the quadcopter, record video, and stream video. The navigation controller is connected directly to the flight controller with wires, however, the smartphone application connects to the navigation controller through Wi-Fi. Wi-Fi was chosen because smartphones have access to a WLAN network card, and will allow for communication over long distances outside. The specific distance that the device can allow, with the embedded Wi-Fi adapter on the raspberry pi, is at least 50 meters. The navigation controller was setup as an access point using WPA security protocol for the smartphone user to connect to before connecting with the application.

Smart Phone Application [Ross Palenik]

The smartphone application is an Android application created using Xamarin with Visual Studio. The language used to implement the application is c#. This language was chosen as Visual Studio is a nice IDE for a new programmer to use, as was the case for this project's smartphone

application developer. The smart phone application is used to send commands to the navigation controller. The first thing the application does on startup is initialize a secure connection between the quadcopter and itself using SSL with a SHA-256 certificate. The user is then required to enter a password that is used to authenticate them. The password that is stored on the quadcopter was saved as a salted hash, so that anyone recovering this saved password will not be able to grab the actual password used.

After transmitting initial variables, the quadcopter will continuously wait to receive any commands from the smartphone application. Also, the smartphone application has another thread created that continuously sends the phones GPS location every two seconds to the navigation controller. The user interface will now be available as well allowing the user to send commands to the quadcopter via the user input part of the flowchart. The user input handler in the smart phone application is broken down into more detailed steps for control of the Hovercam. Figure 21 shows the smart phone application with the user input handler subsystem. This subsystem handles the user inputs that are to be sent to the quadcopter. The Take Off, Hover, and Land inputs each are buttons in the application and each send their respective commands to the quadcopter navigation controller to decipher. The Video Feed, or Video Stream, button takes the user to a separate screen where the application reads a video stream that is being sent from the navigation controller. The Set Distance button takes the user to a screen where a slider can be used to set the desired distance between the smart phone and the quadcopter and that information is sent to the navigation controller. Table 28 below displays the functions used by the Smart Phone Application.

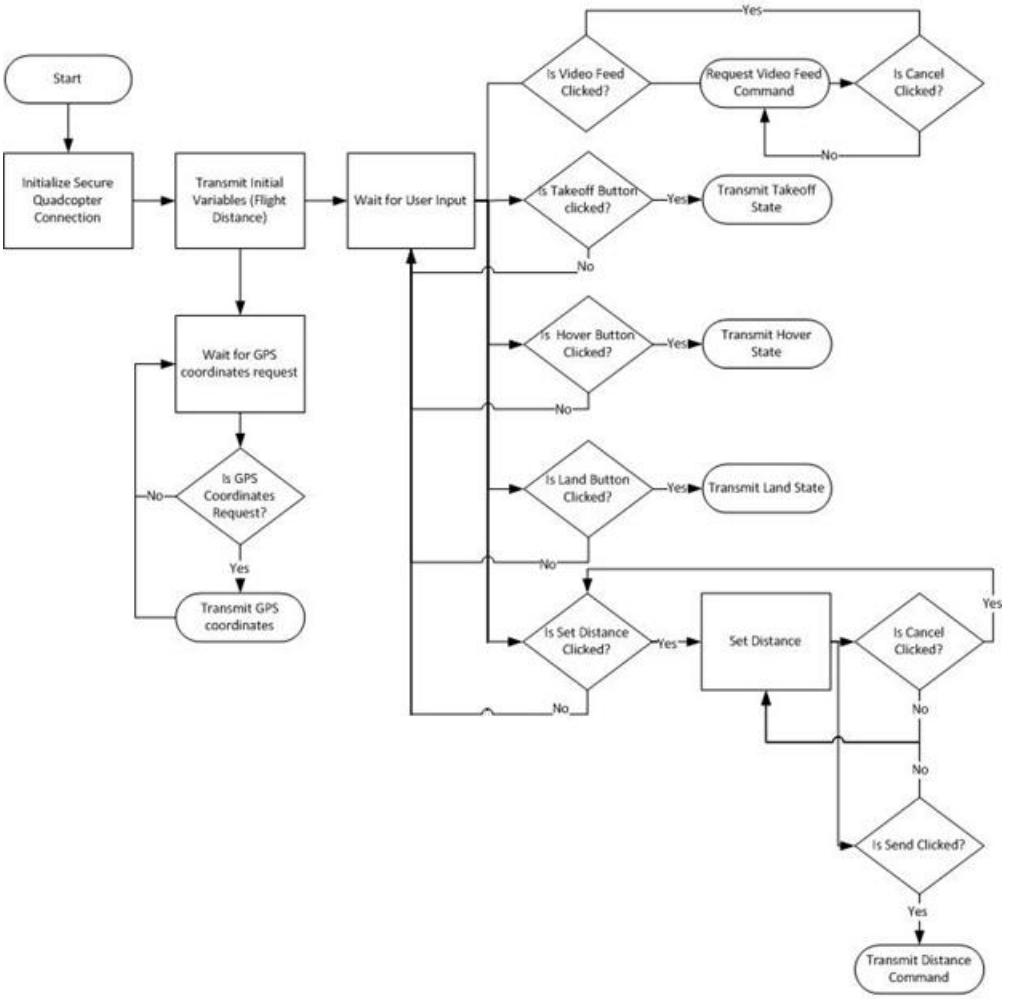


Figure 21: Smart Phone Application Flowchart

Table 28: Smart Phone Application Functions

Function Name	Parameters	Description
IsTakeOffClicked	Command Byte Stream	This function sends takeoff state opcodes to navigation controller.
IsHoverClicked	Command Byte Stream	This function sends hover state opcodes to navigation controller.
IsLandClicked	Command Byte Stream	This function sends land state opcodes to navigation controller.

IsSetDistanceClicked	Command Byte Stream	This function sends set distance state opcodes to navigation controller.
IsGPSCoordinatesRequest	boolean	This function checks if the GPS position has changed, and if so, transmit to the navigation controller.
RequestVideoFeed	Video Byte Stream	This function requests video feed data from the quadcopter microcontroller.

Navigation Controller [Daniel O'Brien]

The navigation controller is the system that communicates with the smartphone application over Wi-Fi and interfaces with the flight controller on the quadcopter. The navigation controller sends flight commands to the flight controller to control the movement of the quadcopter. This process is done using GPS locations to determine where the quadcopter should move. GPS has an accuracy of around 7.8 meters at a 95% confidence. Since this was not the most accurate, image processing was chosen to do fine tracking of the target.

The image processing was implemented using the OpenCV library, which stands for open source computer vision. This was chosen as it is compatible with the chosen Raspberry Pi processor and meets the design requirements of live image processing. The OpenCV library contains many object tracking methods, however, it was determined that the TLD tracking method fit the project parameters best. This is because the TLD tracking method is useful for tracking an unknown object in an unconstrained video stream. This accurately describes the tracking required for this project, as the object to be tracked will be dynamically selected within an unknown space of movement. TLD, tracking-learning-detecting, is a process in which an object is learned, tracked, and detected long-term. Using a medium flow algorithm, a region of interest will be selected as a bounding box in which to learn an object. This object will then be tracked from frame to frame, while the detector localizes all appearances of the object to identify any errors and learn. This will allow for improved tracking to prevent loss of object detection in future frames. While this method was implemented, it was found that it was not accurate enough in a fast enough time frame to autonomously control the quadcopter.

The image processing did not work as was intended as the camera was not on a gimble, as the budget for the project could not support purchasing one, and the target was being lost continuously in a round of testing when the wind would slightly deviate the target from the camera. This cased a stuttering in the following of the target that was not as good as just relying on GPS positioning to follow. This led to the project just depending on GPS coordinates to follow the target, as it was more reliable.

The navigation controller is a multithreaded server created in Python that runs on the Linux distribution, Jessie, for the raspberry pi. Using a Linux based distribution for the operating system allows for less overhead of running the program while still maintaining the functionality to do computation, communicate with the smart phone application, and communicate with the

flight controller. The server itself starts up a subprocess that runs the video streaming and recording. The video streaming and recoding is done at the same time by using raspivid, a native video output application working with the raspberry pi camera, and the Linux bash command ‘tee’. The ‘tee’ command outputs the video coming from camera to a video file and a streamer library called gstreamer. The video file itself is saved as a h264 file format. Gstreamer was chosen to stream video as, out of all the methods tried, it had the lowest streaming latency of around 200-300 ms. The gstreamer application was run using a TCP server listening on a different port than the server, and streams video to any client requesting it.

The server’s main thread uses the TCPServer class to listen for users to connect and send commands shown in Figure 21. This connection is wrapped using the SSL class and a sha-256 certificate to verify the client connecting is valid. Whenever a command is sent, it is parsed and the navigation controller performs one of the valid commands. If the command is takeoff, a new thread is created that autonomously controls and follows the smartphone that initiated the command. This is done by getting the set target GPS position, which was set by using the set GPS coordinates command, and calculating the bearing and distance to the target to move the quadcopter. If the distance is less than 50m, the maximum range set for Wi-Fi, and the minimum distance set using the distance variable, then the quadcopter will move to the target GPS coordinates position. One failsafe implemented is if the Wi-Fi connection to the smartphone application is lost, the quadcopter will return to the position in which it took off from before landing.

The takeoff mode arms the quadcopters motors and waits for them to initialize and arm before attempting to fly. There is a Config file that is used to store any configurable variables as well as global variables set by the target GPs, altitude, and set distance. After the motors are armed and ready to fly, the quadcopter then attempts to reach the altitude set in the Config file. After that altitude is reached, the quadcopter then follows the target. Every two seconds of following the target, the altitude is checked to see if it has deviated from the set altitude in the configuration file, as the wind does blow it off course. If the quadcopter’s altitude has deviated from its set altitude, it adjusts its altitude within five seconds before proceeding to go to a new GPS location.

When the commands to land or hover are sent, the takeoff thread is killed. Then, the quadcopter beings to either land or hover after the thread is killed. Any errors that occur during takeoff or the startup of the server are sent to the smartphone application as long as there is an active connection. These errors or warnings appear on the smartphone application as popup box. Any update messages are sent to the smartphone application’s text field to notify the user of any changes.

Figure 22 shows the user input handler subsystem. This subsystem handles commands sent from the smartphone application. The commands handled are takeoff, land, hover, set distance, and set target GPS position. Each command will have its own opcodes to differentiate between commands. This subsystem also contains another subsystem block, flight controller handler. Table 29 below displays the functions used by the User Input Handler.

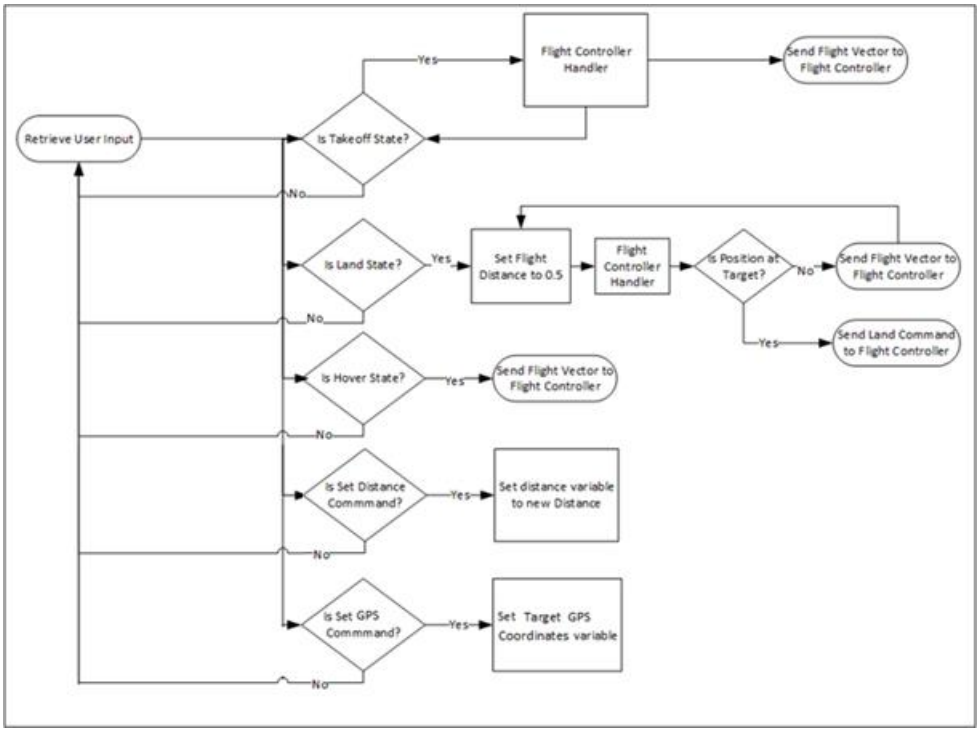


Figure 22: User Input Handler Subsystem

Table 29: User Input Handler Functions

Function Name	Parameters	Description
RetrieveUserInput	Byte stream	This function will handle the incoming byte stream from the application and send it to all of the command state handlers.
IsTakeOffState	Command Byte Stream	This function determines if the bytes sent contain the takeoff state opcodes. It returns a Boolean.
IsTakeOffState	Command Byte Stream	This function determines if the bytes sent contain the takeoff state opcodes. It returns a Boolean.
IsLandSate	Command Byte Stream	This function determines if the bytes sent contain the land state opcodes. It returns a Boolean.
IsSetDistanceCommand	Command Byte Stream	This function determines if the bytes sent contain the set distance state opcodes. It returns a Boolean.
SetFlightDistance	Float	This function sets the distance variable
IsSetGPSCommand	Command Byte Stream	This function sets the target GPS coordinates sent from the smartphone application.

IsPositionAtTarget	None	Determines if the quadcopter is near the target (within 7.8 meters accuracy). It returns a Boolean.
SetRegionOfInterest	Rectangle	This function sets the bounding box of the region of interest for the TLD tracker.

Figure 23 shows the flight controller subsystem. This subsystem flight vector calculation for the flight controller. The GPS coordinates are used to calculate the distance between the quadcopter and the target on the horizontal plane. If the target is within the specified distance, the quadcopter will not move, while if it exceeds the specified distance, the quadcopter will move towards the calculated bearing of the target. Table 30 below displays the functions used by the Flight Controller Handler.

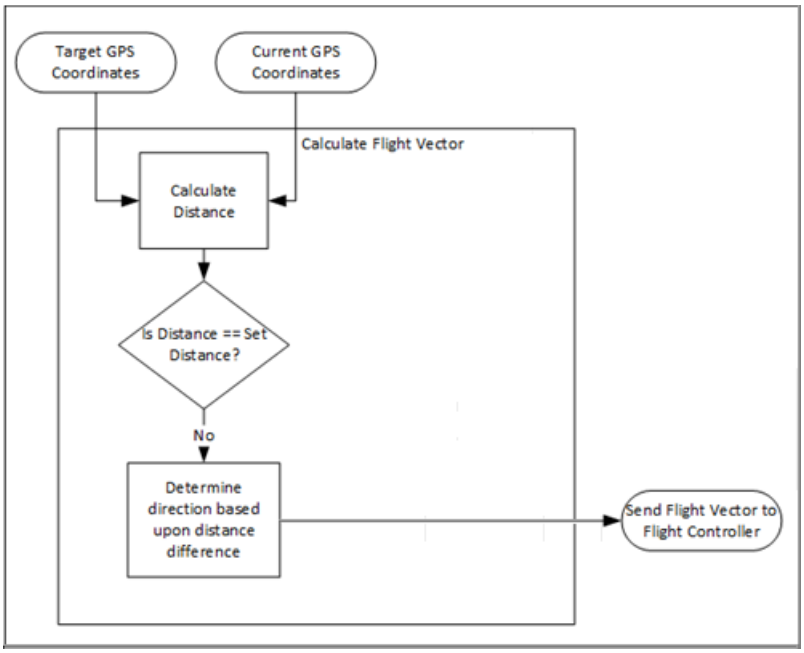


Figure 23: Flight Controller Handler

Table 30: Flight Controller Handler Functions

Function Name	Parameters	Description

CalculateDistance	Target GPS coordinates, quadcopter GPS coordinates	This function determines the difference of the horizontal plane between the target and quadcopter GPS coordinates. This returns the difference as a float.
IsDistanceEqualToSetDistance	Distance	This function determines if the distanced found from calculate distance is equal to the set distance, with a percent of margin error.
DetermineDirectionBased UponDistanceDifference	Distance difference from GPS	This function determines the bearing that the quadcopter must move to maintain a specified distance away from the target.
SendFlightVectorToFlight Controller	Flight vector	This function sends the flight commands to the flight controller to move the quadcopter.

Flight Controller [Kevin Rauh and Daniel O'Brien]

The flight controller monitors the current position and orientation of the Hovercam and compares it to inputs from the navigation controller and the proximity sensors. These compared values are used to control the thrust output of each motor for different states of flight. If an object comes within 2 meters from the Hovercam, detected by the ultrasonic sensors the Hovercam will enter an avoidance state. To avoid the object, the roll of the Hovercam will either increase or decrease depending on which sensor the object is detected on. If the object is detected on either of the right sensors the Hovercam will gain positive roll to fly left of the object. If the object is detected on either of the left sensors the Hovercam will gain negative roll to fly to the right of the object. In either avoidance direction, the pitch of the Hovercam will be adjusted to half of the set value. Once the ultrasonic sensors detect that the object is no longer present, the roll and pitch will return back to the value being sent from the navigation controller. The barometer on the Pixhawk will be used to monitor the height of the Hovercam. The height the Hovercam will be set to maintain is 5 meters. If the height is measured to be too low the thrust of the motors will increase until the correct height is achieved and if the height is measured to be too high the thrust of the motors will decrease until the correct height is achieved. After the correct height is achieved then the thrust of the motors will return back to their original state. Figure 24 below depicts the flow of operation for the flight controller. Table 31 below displays the functions used by the flight controller.

During the actual test of the obstacle avoidance, it was found that the sensors were receiving a lot of false readings. The tests of the ultrasonic sensors by themselves were okay, but the vibration from the quadcopter moving and the actual propellers were causing the ultrasonic sensors to send false readings. This caused the quadcopter to move around randomly when a false reading was received, so it was decided to disable the obstacle avoidance, as it was less of a hazard without it.

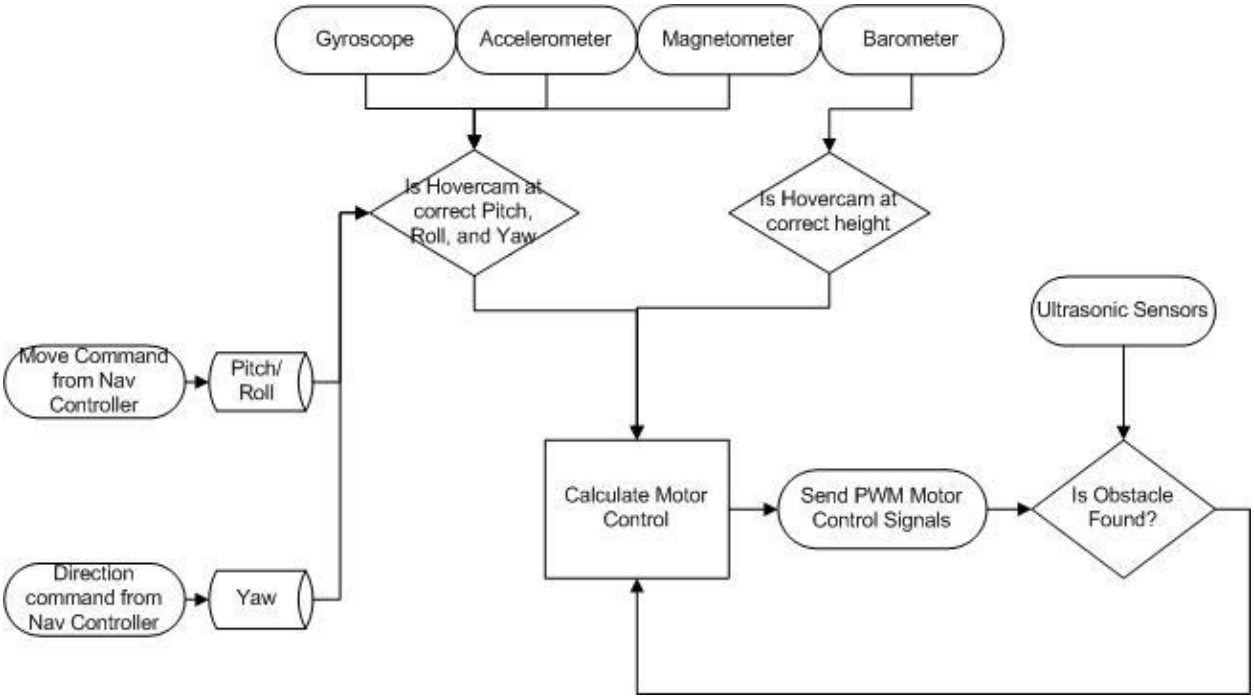


Figure 24: Level 4 Flight Controller Flowchart

Table 31: Flight Controller Functions

Function Name	Parameters	Description
IsObstacleFoundOnLeft	Left side Ultrasonic Sensor above threshold	This function will calculate if there is an object to the left of the Hovercam. Will enter the AvoidLeft function
IsObstacleFoundOnRight	Right side Ultrasonic Sensor above threshold	This function will calculate if there is an object to the right of the Hovercam. Will enter the AvoidRight function
AvoidLeft	IsObstacleFoundOnLeft	If the object is detected on the left sensors the Hovercam will gain negative roll to fly to the right of the object
AvoidRight	IsObstacleFoundOnRight	If the object is detected on the right sensors the Hovercam will gain positive roll to fly to the left of the object
AtHeight	Barometer	If Barometer reads below 5 meters enter GainAltitude function, if

		Barometer reads above 5 meters enter DecreaseAltitude function
GainAltitude	Below 5 meters	Increase all 4 motors output thrust by 10% of maximum thrust
DecreaseAltitude	Above 5 meters	Decrease all 4 motors output thrust by 10% maximum thrust
CheckPitch	Pitch from Gyroscope, Pitch from Navigation controller	Verify the pitch of the Hovercam is at the level sent from the Navigation controller. If they are not the same the pitch is adjusted
CheckRoll	Roll from Gyroscope, Roll from Navigation controller	Verify the roll of the Hovercam is at the level sent from the Navigation controller. If they are not the same the roll is adjusted
CheckYaw	Yaw from Gyroscope, Yaw from Navigation controller	Verify the yaw of the Hovercam is at the level sent from the Navigation controller. If they are not the same the yaw is adjusted

9. Operation, Maintenance, and Repair Instructions

The Hovercam consists of a DJI Flamewheel F450 frame kit with integrated navigation and flight controllers. Since this is an autonomous quadcopter there are a lot of safety hazards and precautions to take into consideration. As stated in the DJI Flamewheel F450 disclaimer "We strongly recommend customers to remove all propellers, use power supply from R/C system or flight pack battery, and keep children away during system calibration and parameter setup. Please respect the AMA's National Model Aircraft Safety Code."

When flying, the fast rotating motors and propellers of the Hovercam can cause serious damage and injury. Therefore, when flying be alert and aware of your surroundings and follow the general guidelines below:

- When flying keep away from objects, such as obstacles, human beings, high-voltage lines, etc.
- Do not get close to or touch the working motors and propellers,
- Do not over load the quadcopter.
- Check whether the propellers and the motors are installed correctly and firmly before flight.
- Make sure the rotation direction of each propeller is correct
- Check whether all parts of quadcopter are in good condition before flight. Do not fly with old or broken parts.

Operation

When operating the Hovercam always make sure you are in an open environment. First attach the propellers to the correct motor, CW or CCW. If the propeller is not on the correct motor the propeller will not screw on and attach firmly. Each propeller, close to the center there will be a small imprint with lock and unlock directional arrows. Once propellers are attached make sure that the micro USB cable is firmly plugged into the Raspberry Pi. Once the propellers are secured and the Raspberry Pi is plugged in you can prepare to launch. First, strap the battery into the Velcro so that the battery is secured. Then, connect the battery deans connector to the receiving end on the Hovercam. After the Hovercam is powered, press and hold the motor arming button on the Hovercam, once the ESCs stop beeping the motors are armed. After arming the motors, connect to the Wi-Fi titled "Hovercam" on your smartphone. Once connected open the Hovercam application, login, and make sure you have a secure connection to the Raspberry Pi. The default password for both the Wi-Fi and login is set to "password". If you wish to change the default passwords, refer to Note 1 below.

After opening the application, give the Hovercam about 15-30 seconds to make a solid connection to the GPS module. Select Take off on the main screen of the application. If everything is done correctly a message "Vehicle is taking off" will appear on the application under the Set Distance button. Once the Hovercam is in the air, proceed to walk away from the Hovercam. Once out of the set following range, the GPS should pick up your new location and proceed towards you. If the Hovercam loses connection the application, it will go into a "Land" sequence and proceed back to a home location at the takeoff position. To make the Hovercam

stay in one position and not follow you, select “Hover” on the smartphone application, the Hovercam will then stay at its current location and height.

To land the Hovercam, select the “Land” command on the application and it will go into a landing sequence. As the Hovercam lands it will touch down on the ground and then rise in the air again briefly before settling down, doing a little “hop”. This hop is so the copter recognizes ground below it before its cuts power to the motors and doesn’t accidently land in the middle of the air if the barometer reading is incorrect. When landing, if the Hovercams propellers are still spinning do not approach. Only proceed to the Hovercam if he propellers have stopped moving. If there are any problems with connecting to the hovercam, please unplug the battery, wait 10 seconds, and plug it back in, allowing the system to reboot. If the issue persists remote into the Raspberry Pi and check what errors are being displayed. The details of how to do this are explained in Note 2.

Note 1: The Wi-Fi password can be changed on the raspberry pi by opening the hostapd configuration file at `/etc/hostapd/hostapd.conf and changing the configuration key wpa_passphrase. The login password can be changed by opening the user file at /home/pi/Documents/Python_Scripts/Server/user.config and changing the configuration key password.`

Note 2: The specific errors are logged into the file /home/pi/Documents/Python_Scripts/Server/hovercam_logs.txt. To remote into the server, you can use Windows Remote Desktop to connect using the static IP 192.168.10.1. The user account id is pi and the password is password. These can be changed after logging into the Linux operating system.

Maintenance and Assembly

After each use remove the propellers from the quadcopter so that they don’t accidently break in storage or in transit. Also, remove the battery after use and recharge it using a charging case so that in the off-chance of a fire the surrounding will be protected. When attaching the propellers make sure they are going on the correct motors. There are two different directional motors on the Hovercam: CW and CCW. There are indications on both the motors and propellers. While installing propellers always make sure the battery is disconnected. Before flight always make sure all the connections are secure and the wires are tied down neatly and out of the way of the spinning propellers. Below in Figure 25 is an assembly diagram for the proper assembly of the DJI Flamewheel F450 frame kit, used for the Hovercam

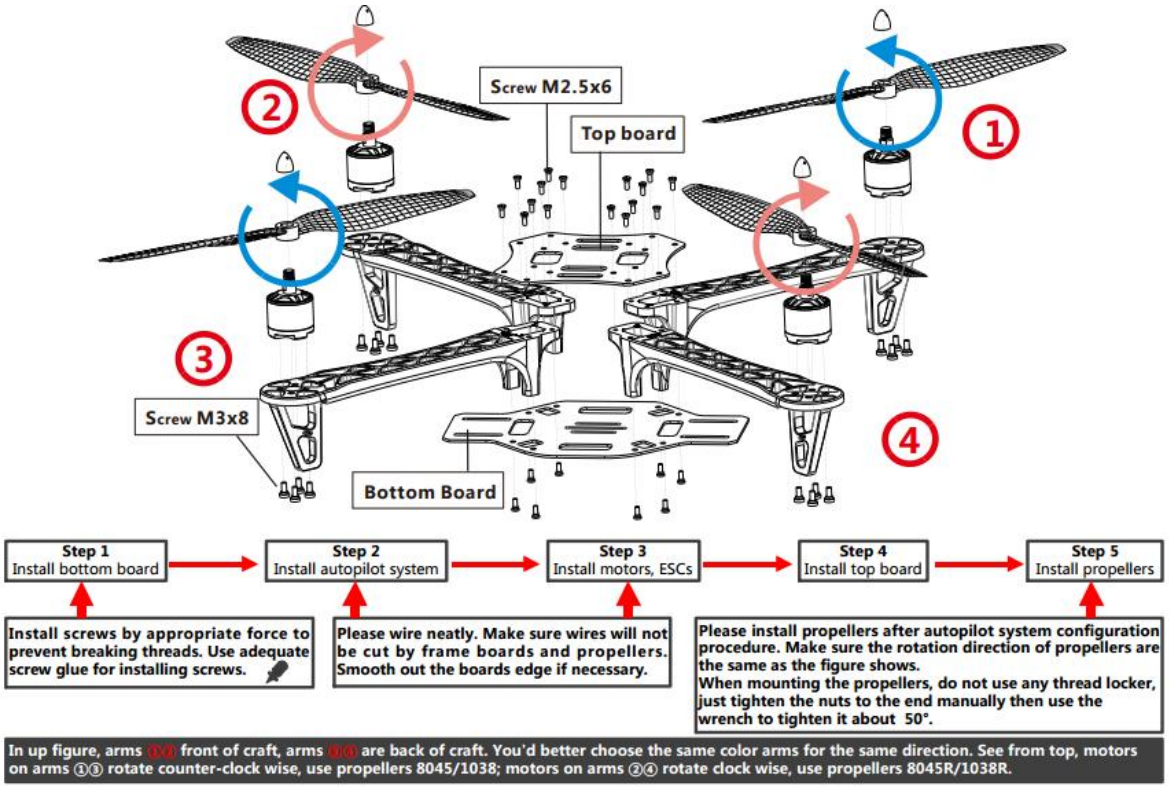


Figure 25: Assembly Diagram for DJI Flamewheel

Repair

When repairing the hovercam the first step is to identify the problem. If a propeller is broken remove it following the arrows on the propeller to unlock it. When replacing a propeller make sure you identify if the propeller is CW or CCW (shown in the assembly diagram above) so you place the correct propeller back on the motor. If a propeller/motor is not spinning correctly first check and make sure all the wires to the ESC are connected correctly. If all the wires are correct, then check the PCB board if the soldering joint failed. If the Hovercam needs disassembled, Figure 26 below depicts the tools needed and a wiring diagram of the ESCs.

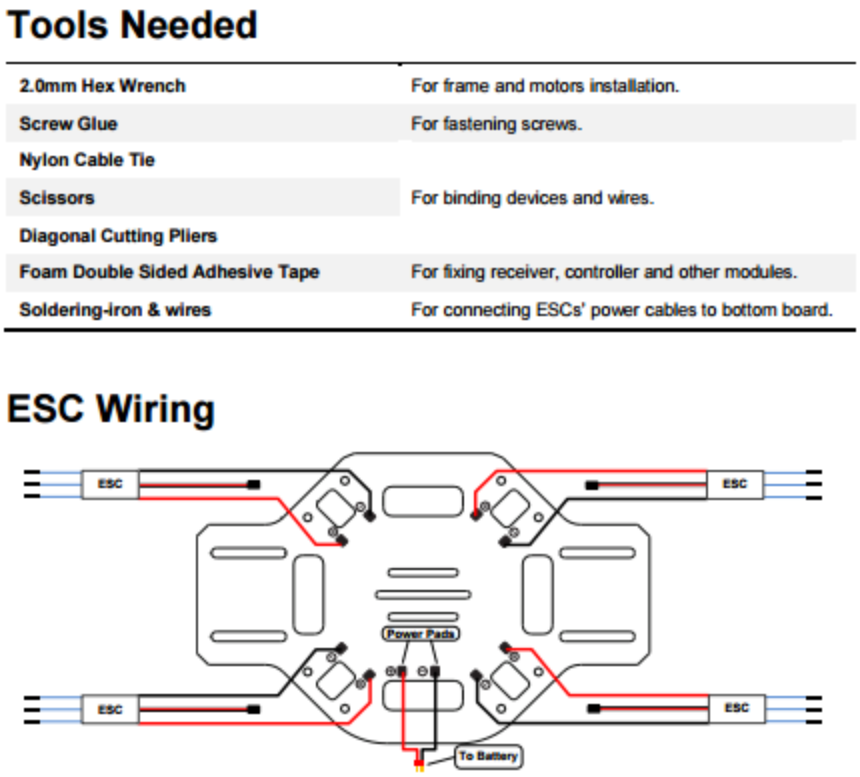


Figure 26: Disassembly Tools

Troubleshooting

If there are any problems starting the Hovercam refer to this document. As the Pixhawk is being powered, a main LED will be flashing with the status of the Hovercam. Below in Table 32 is a list of the different states that the Pixhawk can enter.

Table 32: Pixhawk LED status

LED Status	Meaning
Flashing red and blue	Initializing gyroscopes. Hold the vehicle still and level while it initializes the sensors.
Flashing blue	Disarmed, no GPS lock found. Autopilot, loiter and return-to-launch modes require GPS lock.
Solid blue	Armed with no GPS lock
Flashing green	Disarmed (ready to arm), GPS lock acquired. Quick double tone when disarming from the armed state.

Fast Flashing green	Same as above but GPS is using SBAS (so should have better position estimate).
Solid green	Armed, GPS lock acquired. Ready to fly!
Double flashing yellow	Failing pre-arm checks (system refuses to arm).
Single Flashing yellow	Radio failsafe activated
Flashing yellow (#2)	Battery failsafe activated
Flashing yellow and blue	GPS glitch or GPS failsafe activated
Flashing red and yellow	EKF or Inertial Nav failure
Flashing purple and yellow	Barometer glitch
Solid Red	Error
Solid Red (#2)	SD Card missing (or other SD error like bad format etc.)

Most of the errors that will be encountered are going to be arming issues (Double Flashing Yellow), or a GPS lock/ GPS Fence issue. As stated before, when you start up the Hovercam give about 15-30 seconds to establish a full connection. To fix the GPS lock error, move the Hovercam to an open area to the sky and give the system about 1-5 minutes to establish a GPS connection.

10. Testing Procedures:

The Hovercam has many parts that must be integrated so that the vehicle is operational. Each of the devices was tested before and while they were integrated into a final working quadcopter.

The 5V regulator was first assembled on a breadboard and an input voltage of 12V, the maximum input voltage from the battery, was applied to the regulator. The output and the input of the regulator was attached to an oscilloscope so the voltage levels could be monitored. At every possible voltage level that the battery could supply to the system, from 9V to 12V, a steady 5V was outputted from the regulator circuit.

While testing the maximum distance that the Hovercam could be away from the phone and still stream video feed was tested. In the navigation controls, a maximum of 50m was set, so that if the Hovercam got “lost” or too far away from the user it would return to home, and land. While testing the Hovercam in the field, we exceeded the 50m range set for the Hovercam, and the vehicle returned to land. While it was landing, we were still getting video feed on the cell phone application from the Hovercam. Since we were still able to get video feed while the Hovercam was in a routine for when the vehicle is more than 50m away from the user we know that the communication between the application and the Hovercam can exceed 50m.

To get accurate video feed to the user from the Hovercam there needs to be a low amount of latency between the two systems. To test the latency between the Raspberry Pi and the cell phone application one cell phone was connected to the Raspberry Pi and a second phone had a stopwatch. The two phones were placed side by side and the Hovercam was held in place and recorded the phone with the stopwatch. The recorded image was then displayed on the adjacent phone. In Figure 27 below the latency between the two devices is shown to be between 200 and 300 ms.

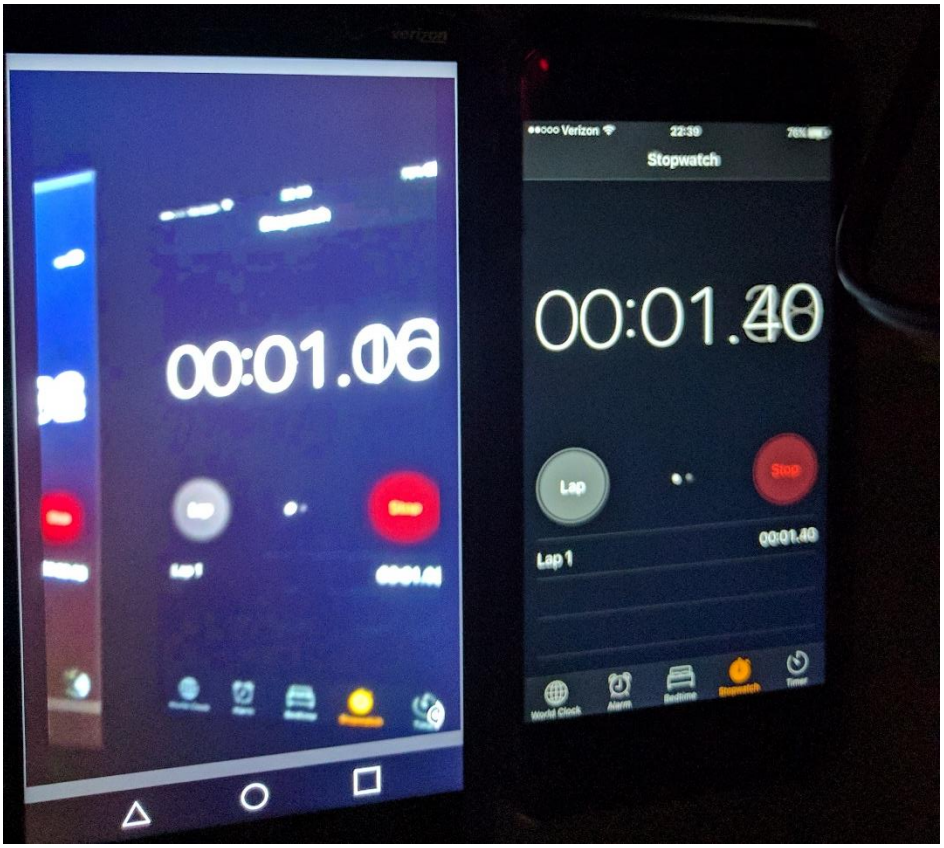


Figure 27: Communication Latency

To test the preliminary flying of the Hovercam, a string was attached to the bottom of the quadcopter and the other end was held by a team member. This string was used so that if the Hovercam started flying towards an object or another person we could restrain the vehicle and avoid any injuries. The string proved to be a good addition to the preliminary testing of the Hovercam, when the navigation controls were first being tested the quadcopter seemed to have a mind of its own and the string was imperative in keeping the Hovercam from getting damaged and from damaging anyone. Figure 28 below shows the string attached to the Hovercam that was used in testing.

Figure 28: Stringed Testing of Hovercam

To test the GPS tracking of the Hovercam we preformed the take-off protocol in the middle of an empty field. We first ran a test to check if the GPS and compass were accurate and calibrated correctly. The Hovercam was instructed to make a square, first by going north, then east, south, and west. After verifying that the compass and GPS were calibrated we entered the standard flight protocol, where the Hovercam would follow GPS coordinates until it was 10m from the target. The user, with the cell phone, then ran around the field and verified that the Hovercam was following them. Figure 29 below is a snapshot from the Hovercam verifying that it was following and recording the user.

Figure 29: 1st Person View Verifying GPS Tracking on Hovercam

11. Parts List and Budget:

The table below shows the projected necessary parts list for the Hovercam. The total estimated cost of the parts came to $471.93. With $400 from the engineering department and $200 from honors this project is within budget, allowing for extra parts if needed in testing.

Table 33: Estimated Budget

Qty.	Part Num.	Description	Unit Cost	Total Cost
1		DJI 450 Frame Kit multi-rotor Wheel ARF kit	165.99	165.99
1		Pixhawk PX4 32Bit ARM Flight Controller	73.99	73.99
1	114990584	Raspberry Pi 3	47.50	47.50
6	SEN-13959	Ultrasonic Sensor 40kHz	3.95	23.70
1	2324	GPS HAT for Raspberry Pi	45.95	45.95
1		Floureon 2 packs 3s2p 8000mAh 40C Lipo Battery	89.99	89.99
1		Raspberry Pi Camera V2 Video module	24.81	24.81
			Total Cost	471.93

As the project progressed additional parts were added to the project. Some of these additional parts added more cost to the budget of the project. Table 34 below depicts the final budget for the Hovercam project.

Table 34: Final Budget

Qty	Part Number	Description	Unit Cost	Total Cost
1		DJI 450 Frame Kit multi-rotor Wheel ARF kit	165.99	165.99

1		Pixhawk PX4 32Bit ARM Flight Controller	73.99	73.99
1	114990584	Raspberry Pi 3	47.50	47.50
6	SEN-13959	Ultrasonic Sensor 40kHz	3.95	23.70
1	2324	GPS HAT for Raspberry Pi	45.95	45.95
1		Floureon 2 packs 3s2p 8000mAh 40C Lipo Battery	89.99	89.99
1		Raspberry Pi Camera V2 Video module	24.81	24.81
1		Male to Female Jumper Wire 40pcs	4.99	4.99
1		GPS and Compass for Pixhawk	33.99	33.99
1		Male Deans Connector 2 pack	5.78	5.78
1		Micro USB cable	26.68	26.68
1		9450 Propellers 4 pairs	13.65	13.65
			Total Cost	557.02

The final cost of the project comes to $557.02, over the $400 that was given to us from the Department. Several of these items were paid for out of pocket so that they would be received in a timely fashion. The added parts for our project were discussed in the Accepted Technical Design section above.

12. Gantt Chart:

Figure 30 below is the Gantt chart layout for the Hovercam design project for the fall of 2016. Figure 31 is the Gantt chart layout for the Hovercam design project for the spring of 2017. Each task is assigned to a start time and a completion time. Leveled breakdowns are included for task to make the completion of the project attainable with the amount of time available.

Name	Begin date	End date
SDP1 fall2016	9/1/16	12/13/16
HoverCam	9/1/16	12/13/16
Preliminary Design Report	9/1/16	9/15/16
Problem Statement	9/1/16	9/15/16
Need	9/1/16	9/8/16
Objective	9/1/16	9/8/16
Background	9/1/16	9/8/16
Marketing Requirements	9/1/16	9/8/16
Objective Tree	9/1/16	9/15/16
Preliminary Design Gantt Chart	9/1/16	9/15/16
Block Diagrams Level 0 w/ FR tables	9/1/16	9/15/16
Hardware modules (identify designer)	9/1/16	9/15/16
Microcontroller (Kevin)	9/1/16	9/15/16
Motors (Adam)	9/1/16	9/15/16
Power	9/1/16	9/15/16
Beacon	9/1/16	9/15/16
Sensors	9/1/16	9/15/16
Video I/O	9/1/16	9/15/16
Software modules (identify designer)	9/1/16	9/15/16
GUI (Ross)	9/1/16	9/15/16
Control (Dan)	9/1/16	9/15/16
Preliminary Design Presentation 3:20PM	9/15/16	9/15/16
Midterm Report	9/15/16	10/16/16
Design Requirements Specification	9/19/16	10/2/16
Midterm Design Gantt Chart	9/19/16	10/2/16
Design Calculations	9/15/16	10/16/16
Electrical Calculations	9/19/16	10/9/16
Preliminary Motor Calculations	9/19/16	10/9/16
Preliminary Hover State Calculations	9/19/16	10/9/16
Power, Voltage, Current	9/19/16	10/9/16
Mechanical Calculations	9/15/16	10/16/16

Structual Considerations	9/15/16	10/16/16
Copter Dimensions and Weight	9/15/16	10/16/16
System Dynamics	9/19/16	10/16/16
Block Diagrams Level 1 w/ FR tables & ToO	9/15/16	10/2/16
Hardware modules (identify designer)	9/15/16	10/2/16
Microcontroller (Ross and Kevin)	9/15/16	10/2/16
Motor Driver (Kevin)	9/15/16	10/2/16
DLDC Motors	9/15/16	10/2/16
Battery	9/15/16	10/2/16
Voltage Regulation (Adam)	9/15/16	10/2/16
Cell Phone	9/15/16	10/2/16
GPS	9/15/16	10/2/16
Communication Signal	9/15/16	10/2/16
Software modules (identify designer)	9/15/16	10/2/16
Smart Phone Application	9/15/16	10/2/16
GUI	9/15/16	10/2/16
Navigation Control (Dan)	9/15/16	10/2/16
Flight Control (Dan)	9/15/16	10/2/16
Block Diagrams Level 2 w/ FR tables & ToO	10/3/16	10/10/16
Hardware modules (identify designer)	10/3/16	10/10/16
Batterys (Adam)	10/3/16	10/10/16
Microcontroller (Kevin)	10/3/16	10/10/16
Sensors	10/3/16	10/10/16
Ultrasonic (Adam)	10/3/16	10/10/16
Flight Controller Sensors (Kevin)	10/3/16	10/10/16
Accelerometer	10/3/16	10/10/16
Gyroscope	10/3/16	10/10/16
Magnetometer	10/3/16	10/10/16
Sonar (Adam)	10/3/16	10/10/16
Navigation Controller (Dan)	10/3/16	10/10/16
Wifi Adapter	10/3/16	10/10/16
Camera	10/3/16	10/10/16
GPS Sensor	10/3/16	10/10/16
Smart Phone	10/3/16	10/10/16
Wifi Adapter	10/3/16	10/10/16
Motors	10/3/16	10/10/16

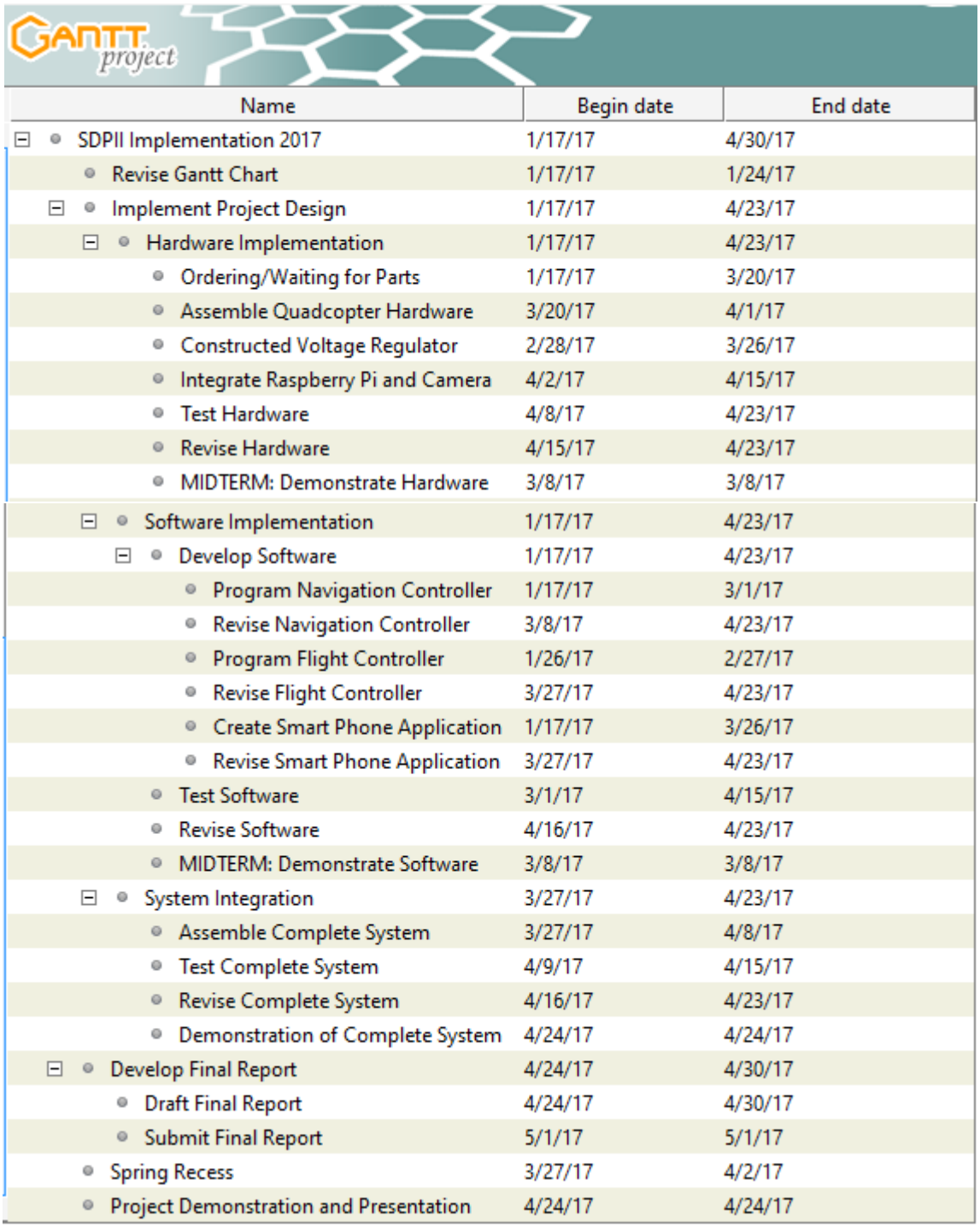
ESCs (Ross)	10/3/16	10/10/16
Software modules (identify designer)	10/3/16	10/10/16
Smart Phone Application (Ross)	10/3/16	10/10/16
Flow Chart	10/3/16	10/10/16
Basic GUI Design	10/3/16	10/10/16
Navigation Controller (Dan)	10/3/16	10/10/16
Flow Chart	10/3/16	10/10/16
Flight Controller (Dan)	10/3/16	10/10/16
Flow Chart	10/3/16	10/10/16
Midterm Design Presentations 3:20-5:00PM Part 1	10/20/16	10/20/16
Midterm Design Presentations 3:20-5:00PM Part 2	10/27/16	10/27/16
Project Poster	10/27/16	11/7/16
Final Design Report	10/17/16	12/2/16
Abstract, Need, and Objective	10/17/16	11/30/16
Background	10/17/16	11/30/16
Motors	10/17/16	11/30/16
Battery	10/17/16	11/30/16
Flight Controller	10/17/16	11/30/16
Smart Phone Application	10/17/16	11/30/16
Navigation Controller	10/17/16	11/30/16
Design and Marketing Requirements	10/17/16	11/30/16
Software Design	10/17/16	12/2/16
GUI (Ross)	10/17/16	11/30/16
Flowcharts	10/17/16	11/30/16
Main	10/17/16	11/30/16
User Input Handler	10/17/16	11/30/16
Function Table	10/17/16	11/30/16
GUI Design	10/17/16	11/30/16
Navigation Controller (Dan)	10/19/16	12/2/16
Flowcharts	10/19/16	12/2/16
Main	10/19/16	12/2/16
Battery Life Check	10/19/16	12/2/16
User Input Handler	10/19/16	12/2/16
Flight Controller Handler	10/19/16	12/2/16

Fuction Table	10/19/16	12/2/16
Main	10/19/16	12/2/16
Battery Life Check	10/19/16	12/2/16
User Input Hadler	10/19/16	12/2/16
Flight Controller Handler	10/19/16	12/2/16
Flight Controller (Kevin)	10/19/16	12/2/16
Flowchart	10/19/16	12/2/16
Function Table	10/19/16	12/2/16
Sensor Information	10/19/16	12/2/16
Hardware Design (Adam and Kevin)	10/17/16	11/30/16
Quadcopter Design	10/17/16	11/30/16
Block Diagrams	10/17/16	11/30/16
Motor and Propeller Calculations	10/17/16	11/30/16
Thrust Calculations	10/17/16	11/30/16
Battery Calculations	10/17/16	11/30/16
Budget (Estimated)	10/19/16	12/2/16
Final Required Parts List	10/19/16	12/2/16
Implementation Gantt Chart	10/17/16	11/30/16
Conclusions and Recommendations	10/17/16	11/30/16
Final Design Presentation Part 1 3:20PM-5:00PM	12/1/16	12/1/16
Final Design Presentation Part 2 3:20PM-5:00PM	12/8/16	12/8/16
Final Design Presentation Part 3 5:15PM-7:15PM	12/14/16	12/14/16

Figure 30: Fall Semester Gant Chart

Name	Begin date	End date
SDPII Implementation 2017	1/17/17	4/30/17
Revise Gantt Chart	1/17/17	1/24/17
Implement Project Design	1/17/17	4/16/17
Hardware Implementation	1/17/17	3/10/17
Assemble Quadcopter Hardware	1/17/17	1/29/17
Integrate Raspberry Pi and Camera	1/30/17	2/12/17
Integrate Sensors	2/13/17	2/19/17
Test Hardware	2/20/17	3/5/17
Revise Hardware	2/20/17	3/5/17
MIDTERM: Demonstrate Hardware	3/6/17	3/10/17
SDC & FA Hardware Approval	3/11/17	3/11/17
Software Implementation	1/17/17	3/10/17
Develop Software	1/17/17	2/12/17
Program Navigation Controller	1/17/17	2/12/17
Program Flight Controller	1/17/17	2/12/17
Create Smart Phone GUI	1/17/17	2/12/17
Test Software	2/13/17	3/5/17
Revise Software	2/13/17	3/5/17
MIDTERM: Demonstrate Software	3/6/17	3/10/17
SDC & FA Software Approval	3/11/17	3/11/17
System Integration	3/12/17	4/16/17
Assemble Complete System	3/12/17	3/26/17
Test Complete System	3/27/17	4/16/17
Revise Complete System	3/27/17	4/16/17
Demonstration of Complete System	4/17/17	4/17/17
Develop Final Report	1/17/17	4/30/17
Write Final Report	1/17/17	4/30/17
Submit Final Report	5/1/17	5/1/17
Spring Recess	3/27/17	4/2/17
Project Demonstration and Presentation	4/24/17	4/24/17

Figure 31: Spring 2017 Tentative Gant Chart



Name	Begin date	End date
SDPII Implementation 2017	1/17/17	4/30/17
Revise Gantt Chart	1/17/17	1/24/17
Implement Project Design	1/17/17	4/23/17
Hardware Implementation	1/17/17	4/23/17
Ordering/Waiting for Parts	1/17/17	3/20/17
Assemble Quadcopter Hardware	3/20/17	4/1/17
Constructed Voltage Regulator	2/28/17	3/26/17
Integrate Raspberry Pi and Camera	4/2/17	4/15/17
Test Hardware	4/8/17	4/23/17
Revise Hardware	4/15/17	4/23/17
MIDTERM: Demonstrate Hardware	3/8/17	3/8/17
Software Implementation	1/17/17	4/23/17
Develop Software	1/17/17	4/23/17
Program Navigation Controller	1/17/17	3/1/17
Revise Navigation Controller	3/8/17	4/23/17
Program Flight Controller	1/26/17	2/27/17
Revise Flight Controller	3/27/17	4/23/17
Create Smart Phone Application	1/17/17	3/26/17
Revise Smart Phone Application	3/27/17	4/23/17
Test Software	3/1/17	4/15/17
Revise Software	4/16/17	4/23/17
MIDTERM: Demonstrate Software	3/8/17	3/8/17
System Integration	3/27/17	4/23/17
Assemble Complete System	3/27/17	4/8/17
Test Complete System	4/9/17	4/15/17
Revise Complete System	4/16/17	4/23/17
Demonstration of Complete System	4/24/17	4/24/17
Develop Final Report	4/24/17	4/30/17
Draft Final Report	4/24/17	4/30/17
Submit Final Report	5/1/17	5/1/17
Spring Recess	3/27/17	4/2/17
Project Demonstration and Presentation	4/24/17	4/24/17

Figure 32: Spring 2017 Final Gant Chart

The Gantt chart for the 2017 Spring semester saw some changes from its original iteration. The major reason for these changes is due to the amount of time it took to get the parts ordered and delivered. This pushed back much of the hardware construction which, in turn, delayed software testing necessary for furthering the implemented software. Revisions for each

of the controllers and application were made very late in the timeline and had to be done quickly since so many delays in the hardware and testing compounded.

13. Design Team Information

Kevin Rauh, Electrical Engineering, Controls Systems II, Programmable Logic, Embedded Systems Interfacing, Active Circuits, VLSI Design, Electromagnetic Compatibility

Ross Palenik, Electrical Engineering, Controls Systems II, Programmable Logic, Embedded Systems Interfacing, Electromagnetic Compatibility, Active Circuits, VLSI Design

Adam Fada, Electrical Engineering, Digital Communications, Embedded Systems Interfacing, Analog IC Design, VLSI Design, Electromagnetic Compatibility, Antenna Theory.

Daniel O'Brien, Computer Engineering, Data Structures, Programmable Logic, Embedded Scientific Computing, Algorithms, Object-Oriented Programming

14. Conclusions and Recommendations

When building a quadcopter, two things to take into consideration are weight and power consumption. The total weight of the quadcopter should include everything: frame, controllers, PCB, wires, motors, ESC's, battery, and payload, such as a HD camera. By knowing the size of the frame, the maximum propeller size can be determined. Once these two things have been chosen, the thrust of the motors can now be roughly calculated to deliver enough lift for the Hovercam to achieve flight. A general rule is that at least twice as much thrust than the weight of the quadcopter should be provided. This is the bare minimum to ensure you have a stable copter that is easy to control in a hover state. If the thrust provided by the motors is too little, the quadcopter would not respond well to control, it might even have difficulties taking off. For example, if a quadcopter weights 1.5kg, like the Hovercam, the total thrust generated by the motors at 100% throttle should be at least 3kg, or 750g per motor, for a quadcopter. For faster flying drones, like drone racing, the ratio should be much higher than 2:1. It's not uncommon to see a mini quad with a thrust to weight ratio of 8:1 or even 10:1. The quadcopter's performance is much more agile and dynamic with this higher ratio, it accelerates faster and corners better. When there is an excessively high thrust to weight ratio, the quadcopter becomes hard to control since a little increase in throttle is enough to "shoot the quad into orbit". It is recommended to build quadcopter between a 3:1 and 4:1 ratio, like the Hovercam, if it is meant to fly as a slow aerial photography platform. This lower ratio gives better controllability, but also the room for adding extra payload in the future, such as heavier cameras or extra batteries to extend flight time. After choosing a frame size, appropriate motors were chosen. The frame size limits propeller size and propeller size limits motor size. The motor thrust output test data was used to determine the maximum thrust of the motor, when pairing the chosen propellers. The decision to use 1045 propellers, which was within the constraints of the frame size, let the team opt for the motors that were chosen. It is also important to understand that voltage had a large impact on the motor and propeller choice. Motors try to spin much harder when a higher voltage is applied, and thus draw a higher current. When choosing the best motor, the following factors were considered: thrust, current draw, efficiency, and weight. When using less efficient motors not only is a lot of energy wasted and flight time lost, but the maximum possible thrust will decrease. When a motor runs inefficiently, the response time of the quadcopter suffers. It will take the motors longer to reach desired RPM and this will have negative influence on stability and responsiveness. Higher thrust results in high speeds, but changes the efficiency. It was necessary to make sure the selected ESCs had a higher current rating than the motors. The choice of motors and propellers affected which battery was used. If the quadcopter draws a large amount of current at full throttle, the battery's max discharge rate must be able to provide that amount of current continuously. The same applies to brushless motor, the higher efficiency the better. A 70% efficient motor produces 70% power and 30% heat. A 90% efficient motor produces 90% power and 10% heat. After considering all the information above, appropriate motors, ESC's, propellers, batteries, and an appropriate frame were chosen to meet the requirements of the project.

The software design of the Hovercam meets most of the design requirements. The choice of using an android smartphone was a cost-effective decision as the majority of people who would use the Hovercam would have a smartphone. Using Wi-Fi for the connection between the smartphone application and the navigation controller is more than adequate for the specified range requirement of 30m, but a stronger Wi-Fi module than one the team had to use due to budgetary constraints would have made the Hovercam more reliable and capable of longer range communication. Using a Raspberry Pi 3 Model B for the navigation controller was adequate to perform as required for live streaming the recorded footage to the smartphone and for storing a copy on itself. The design of the smartphone application worked well and was not too hard to create, but creating a way for the application to communicate with the hardware was quite challenging. Writing the software for the navigation controller and for the application in the same computer language is recommended. The idea to use proximity sensors for obstacle avoidance worked well on paper as the sensors performed perfectly when tested in the lab, but utterly failed in practice as they returned obstacles when there were none. It is believed that the sensors were picking up the movement of the propellers which would cause this issue. A potential fix would be to construct mounts for the sensors that are angled down as to not allow an potential for the propellers to cross into the range of the sensors. The design of using a navigation controller with image processing to control the yaw of the quadcopter turned out to be a more difficult task than anticipated when trying to handle everything else the team was tasked with. The success of using GPS to control the direction of the quadcopter as it followed the smartphone user was quite an accomplishment in itself. Stabilization became a little bit of a problem as the testing period went on and the video feedback would become relatively shaky at some points. It would be recommended to perform hardware stabilization of the camera such as adding a gimbal instead of software as that would require less processing for the Raspberry Pi 3.

15. References

Y. Colin, B. Bertrand, A. Patron, V. Pho, “Skyteboard quadcopter and method”, U.S. Patent US9004396 B1, April 14, 2015.

L. Gang, L.S Wei, “Method and system for unmanned plane to conduct vehicle tracking”, Patent CN104766481 A, June 8, 2015.

H.L.M. Chow, C.C.J. Chung, K.Y.C. Lai, “Personal tracking device with low power consumption”, U.S. Patent US8195192 B2, June 5, 2012

Kalal, K. Mikolajczyk, and J. Matas, “Tracking-Learning- Detection,” Pattern Analysis and Machine Intelligence 2011. <http://kahlan.eps.surrey.ac.uk/featurespace/tld/Publications/2011_tpami>

http://www.gps.gov/technical/ps/2008-SPS- performance-standard.pdf

"Flamewheel F450 User Manual". http://download.dji-innovations.com/downloads/flamewheel/en/F450_User_Manual_v2.1_en.pdf. N.p., 2017. Web. 30 Apr. 2017.

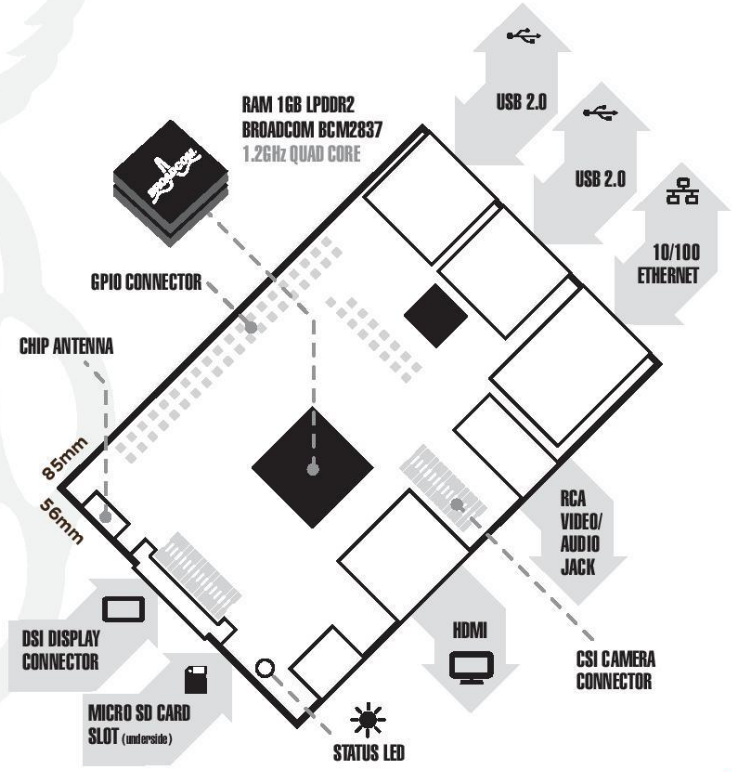
16. Appendix:

Raspberry Pi 3 model B

Raspberry Pi

Raspberry Pi 3 Model B

Product Name	**Raspberry Pi 3**
Product Description	The Raspberry Pi 3 Model B is the third generation Raspberry Pi. This powerful credit-card sized single board computer can be used for many applications and supersedes the original Raspberry Pi Model B+ and Raspberry Pi 2 Model B. Whilst maintaining the popular board format the Raspberry Pi 3 Model B brings you a more powerful processer, 10x faster than the first generation Raspberry Pi. Additionally it adds wireless LAN & Bluetooth connectivity making it the ideal solution for powerful connected designs.
RS Part Number	**896-8660**



RS

www.rs-components.com/raspberrypi

Raspberry Pi

Raspberry Pi 3 Model B

Specifications

Processor	Broadcom BCM2387 chipset. 1.2GHz Quad-Core ARM Cortex-A53 802.11 b/g/n Wireless LAN and Bluetooth 4.1 (Bluetooth Classic and LE)
GPU	Dual Core VideoCore IV® Multimedia Co-Processor. Provides Open GL ES 2.0, hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode. Capable of 1Gpixel/s, 1.5Gtexel/s or 24GFLOPs with texture filtering and DMA infrastructure
Memory	1GB LPDDR2
Operating System	Boots from Micro SD card, running a version of the Linux operating system or Windows 10 IoT
Dimensions	85 x 56 x 17mm
Power	Micro USB socket 5V1, 2.5A

Connectors:

Ethernet	10/100 BaseT Ethernet socket
Video Output	HDMI (rev 1.3 & 1.4 Composite RCA (PAL and NTSC)
Audio Output	Audio Output 3.5mm jack, HDMI USB 4 x USB 2.0 Connector
GPIO Connector	40-pin 2.54 mm (100 mil) expansion header: 2x20 strip Providing 27 GPIO pins as well as +3.3 V, +5 V and GND supply lines
Camera Connector	15-pin MIPI Camera Serial Interface (CSI-2)
Display Connector	Display Serial Interface (DSI) 15 way flat flex cable connector with two data lanes and a clock lane
Memory Card Slot	Push/pull Micro SDIO

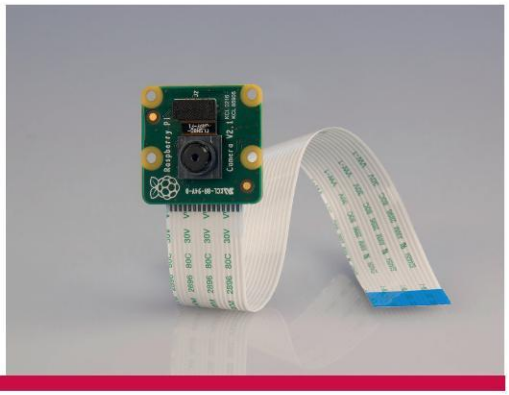
Key Benefits

- Low cost
- 10x faster processing
- Consistent board format
- Added connectivity

Key Applications

- Low cost PC/tablet/laptop
- Media centre
- Industrial/Home automation
- Print server
- Web camera
- Wireless access point
- IoT applications
- Robotics
- Server/cloud server
- Security monitoring
- Gaming
- Environmental sensing/monitoring (e.g. weather station)

Raspberry Pi



Camera Module

Product Name	**Raspberry Pi Camera Module**
Product Description	High Definition camera module compatible with all Raspberry Pi models. Provides high sensitivity, low crosstalk and low noise image capture in an ultra small and lightweight design. The camera module connects to the Raspberry Pi board via the CSI connector designed specifically for interfacing to cameras. The CSI bus is capable of extremely high data rates, and it exclusively carries pixel data to the processor.
RS Part Numer	**913-2664**

Specifications

Image Sensor	Sony IMX 219 PQ CMOS image sensor in a fixed-focus module.
Resolution	8-megapixel
Still picture resolution	3280 x 2464
Max image transfer rate	1080p: 30fps (encode and decode) 720p: 60fps
Connection to Raspberry Pi	15-pin ribbon cable, to the dedicated 15-pin MIPI Camera Serial Interface (CSI-2).
Image control functions	Automatic exposure control Automatic white balance Automatic band filter Automatic 50/60 Hz luminance detection Automatic black level calibration
Temp range	Operating: -20° to 60° Stable image: -20° to 60°
Lens size	1/4"
Dimensions	23.86 x 25 x 9mm
Weight	3g

Pixhawk

RB-Trs-32
3DR PX4 Pixhawk Advanced Autopilot

Pixhawk is an advanced autopilot system designed by the PX4 open-hardware project and manufactured by 3D Robotics. It features advanced processor and sensor technology from ST Microelectronics® and a NuttX real-time operating system, delivering incredible performance, flexibility, and reliability for controlling any autonomous vehicle.

The benefits of the Pixhawk system include integrated multithreading, a Unix/Linux-like programming environment, completely new autopilot functions such as Lua scripting of missions and flight behavior, and a custom PX4 driver layer ensuring tight timing across all processes. These advanced capabilities ensure that there are no limitations to your autonomous vehicle. Pixhawk allows existing APM and PX4 operators to seamlessly transition to this system and lowers the barriers to entry for new users to participate in the exciting world of autonomous vehicles.

The flagship Pixhawk module will be accompanied by new peripheral options, including a digital airspeed sensor, support for an external multi-color LED indicator and an external magnetometer. All peripherals are automatically detected and configured.

Features

- Advanced 32 bit ARM Cortex® M4 Processor running NuttX RTOS
- 14 PWM/servo outputs (8 with failsafe and manual override, 6 auxiliary, high-power compatible)
- Abundant connectivity options for additional peripherals (UART, I2C, CAN)
- Integrated backup system for in-flight recovery and manual override with dedicated processor and stand-alone power supply
- Backup system integrates mixing, providing consistent autopilot and manual override mixing modes
- Redundant power supply inputs and automatic failover
- External safety button for easy motor activation

- Multicolor LED indicator
- High-power, multi-tone piezo audio indicator
- microSD card for long-time high-rate logging

Specifications

Microprocessor
- 32 bit STM32F427 Cortex M4 core with FPU
- 168 MHz/256 KB RAM/2 MB Flash
- 32 bit STM32F103 failsafe co-processor

Sensors
- ST Micro L3GD20H 16 bit gyroscope
- ST Micro LSM303D 14 bit accelerometer / magnetometer
- MEAS MS5611 barometer

Interfaces
- 5x UART (serial ports), one high-power capable, 2x with HW flow control
- 2x CAN
- Spektrum DSM / DSM2 / DSM-X® Satellite compatible input
- Futaba S.BUS® compatible input and output
- PPM sum signal
- RSSI (PWM or voltage) input
- I2C®
- SPI
- 3.3 and 6.6V ADC inputs
- External microUSB port

Power System
- Ideal diode controller with automatic failover
- Servo rail high-power (7 V) and high-current ready
- All peripheral outputs over-current protected, all inputs ESD protected

Weight and Dimensions
- Weight: 38g (1.31oz)
- Width: 50mm (1.96")
- Thickness: 15.5mm (.613")
- Length: 81.5mm (3.21")

Voltage Regulator

TEXAS INSTRUMENTS

LM2576, LM2576HV

SNVS107D – JUNE 1999 – REVISED MAY 2016

LM2576xx Series SIMPLE SWITCHER® 3-A Step-Down Voltage Regulator

1 Features

- 3.3-V, 5-V, 12-V, 15-V, and Adjustable Output Versions
- Adjustable Version Output Voltage Range,1.23 V to 37 V (57 V for HV Version) ±4% Maximum Over Line and Load Conditions
- Specified 3-A Output Current
- Wide Input Voltage Range: 40 V Up to 60 V for HV Version
- Requires Only 4 External Components
- 52-kHz Fixed-Frequency Internal Oscillator
- TTL-Shutdown Capability, Low-Power Standby Mode
- High Efficiency
- Uses Readily Available Standard Inductors
- Thermal Shutdown and Current Limit Protection

2 Applications

- Simple High-Efficiency Step-Down (Buck) Regulator
- Efficient Preregulator for Linear Regulators
- On-Card Switching Regulators
- Positive-to-Negative Converter (Buck-Boost)

3 Description

The LM2576 series of regulators are monolithic integrated circuits that provide all the active functions for a step-down (buck) switching regulator, capable of driving 3-A load with excellent line and load regulation. These devices are available in fixed output voltages of 3.3 V, 5 V, 12 V, 15 V, and an adjustable output version.

Requiring a minimum number of external components, these regulators are simple to use and include fault protection and a fixed-frequency oscillator.

The LM2576 series offers a high-efficiency replacement for popular three-terminal linear regulators. It substantially reduces the size of the heat sink, and in some cases no heat sink is required.

A standard series of inductors optimized for use with the LM2576 are available from several different manufacturers. This feature greatly simplifies the design of switch-mode power supplies.

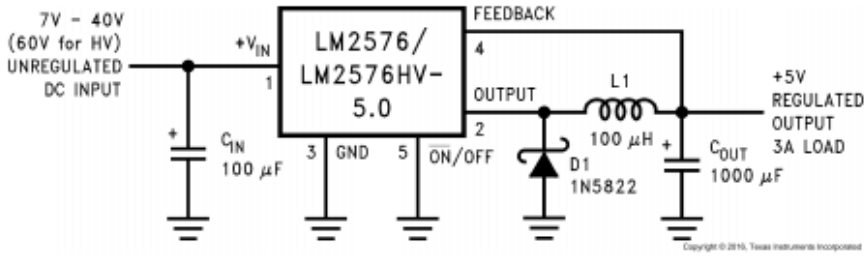
Other features include a ±4% tolerance on output voltage within specified input voltages and output load conditions, and ±10% on the oscillator frequency. External shutdown is included, featuring 50-µA (typical) standby current. The output switch includes cycle-by-cycle current limiting, as well as thermal shutdown for full protection under fault conditions.

Device Information(1)

PART NUMBER	PACKAGE	BODY SIZE (NOM)
LM2576 LM2576HV	TO-220 (5)	10.16 mm × 8.51 mm
	DDPAK/TO-263 (5)	10.16 mm × 8.42 mm

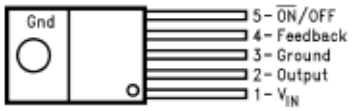
(1) For all available packages, see the orderable addendum at the end of the data sheet.

Fixed Output Voltage Version Typical Application Diagram

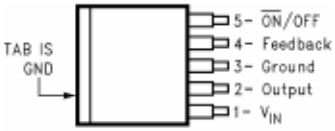


5 Pin Configuration and Functions

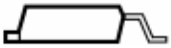
KC Package
5-Pin TO-220
Top View



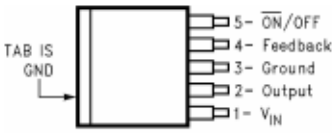
KTT Package
5-PIN DDPAK/TO-263
Top View



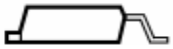
Side View



DDPAK/TO-263 (S) Package
5-Lead Surface-Mount Package
Top View



Side View



Pin Functions

PIN		I/O(1)	DESCRIPTION
NO.	NAME		
1	V_{IN}	I	Supply input pin to collector pin of high-side transistor. Connect to power supply and input bypass capacitors CIN. Path from V_{IN} pin to high frequency bypass C_{IN} and GND must be as short as possible.
2	OUTPUT	O	Emitter pin of the power transistor. This is a switching node. Attached this pin to an inductor and the cathode of the external diode.
3	GROUND	—	Ground pin. Path to C_{IN} must be as short as possible.
4	FEEDBACK	I	Feedback sense input pin. Connect to the midpoint of feedback divider to set VOUT for ADJ version or connect this pin directly to the output capacitor for a fixed output version.
5	$\overline{ON}$/OFF	I	Enable input to the voltage regulator. High = OFF and low = ON. Connect to GND to enable the voltage regulator. Do not leave this pin float.
—	TAB	—	Connected to GND. Attached to heatsink for thermal relief for TO-220 package or put a copper plane connected to this pin as a thermal relief for DDPAK package.

(1) I = INPUT, O = OUTPUT

6 Specifications

6.1 Absolute Maximum Ratings

over the recommended operating junction temperature range of -40°C to 125°C (unless otherwise noted)[1][2]

		MIN	MAX	UNIT
Maximum supply voltage	LM2576		45	V
	LM2576HV		63	
$\overline{ON}$ /OFF pin input voltage		−0.3V ≤ V ≤ +V_{IN}		V
Output voltage to ground	(Steady-state)	−1		V
Power dissipation		Internally Limited		
Maximum junction temperature, T_J			150	°C
Storage temperature, T_{stg}		−65	150	°C

(1) Stresses beyond those listed under *Absolute Maximum Ratings* may cause permanent damage to the device. These are stress ratings only, which do not imply functional operation of the device at these or any other conditions beyond those indicated under *Recommended Operating Conditions*. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.
(2) If Military/Aerospace specified devices are required, please contact the TI Sales Office/ Distributors for availability and specifications.

6.2 ESD Ratings

			VALUE	UNIT
$V_{(ESD)}$	Electrostatic discharge	Human-body model (HBM), per ANSI/ESDA/JEDEC JS-001[1]	±2000	V

(1) JEDEC document JEP155 states that 500-V HBM allows safe manufacturing with a standard ESD control process.

6.3 Recommended Operating Conditions

over the recommended operating junction temperature range of -40°C to 125°C (unless otherwise noted)

		MIN	MAX	UNIT
Temperature	LM2576, LM2576HV	−40	125	°C
Supply voltage	LM2576		40	V
	LM2576HV		60	

6.4 Thermal Information

THERMAL METRIC[1][2][3]		LM2576, LM2576HV		UNIT
		KTT (TO-263)	KC (TO-220)	
		5 PINS	5 PINS	
$R_{\theta JA}$	Junction-to-ambient thermal resistance	42.6	32.4	°C/W
$R_{\theta JC(top)}$	Junction-to-case (top) thermal resistance	43.3	41.2	°C/W
$R_{\theta JB}$	Junction-to-board thermal resistance	22.4	17.6	°C/W
ψ_{JT}	Junction-to-top characterization parameter	10.7	7.8	°C/W
ψ_{JB}	Junction-to-board characterization parameter	21.3	17	°C/W
$R_{\theta JC(bot)}$	Junction-to-case (bottom) thermal resistance	0.4	0.4	°C/W

(1) For more information about traditional and new thermal metrics, see the *Semiconductor and IC Package Thermal Metrics* application report, SPRA953 and the *Using New Thermal Metrics* applications report, SBVA025.
(2) The package thermal impedance is calculated in accordance with JESD 51-7
(3) Thermal Resistances were simulated on a 4-layer, JEDEC board.

6.5 Electrical Characteristics: 3.3 V

Specifications are for T_J = 25°C (unless otherwise noted).

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
SYSTEM PARAMETERS TEST CIRCUIT Figure 26 and Figure 32(1)							
V_{OUT}	Output Voltage	V_{IN} = 12 V, I_{LOAD} = 0.5 A Circuit of Figure 26 and Figure 32		3.234	3.3	3.366	V
	Output Voltage: LM2576	6 V ≤ V_{IN} ≤ 40 V, 0.5 A ≤ I_{LOAD} ≤ 3 A Circuit of Figure 26 and Figure 32	T_J = 25°C	3.168	3.3	3.432	V
			Applies over full operating temperature range	3.135		3.465	
	Output Voltage: LM2576HV	6 V ≤ V_{IN} ≤ 60 V, 0.5 A ≤ I_{LOAD} ≤ 3 A Circuit of Figure 26 and Figure 32	T_J = 25°C	3.168	3.3	3.45	V
			Applies over full operating temperature range	3.135		3.482	
η	Efficiency	V_{IN} = 12 V, I_{LOAD} = 3 A			75%		

(1) External components such as the catch diode, inductor, input and output capacitors can affect switching regulator system performance. When the LM2576/LM2576HV is used as shown in Figure 26 and Figure 32, system performance is as shown in *Electrical Characteristics: All Output Voltage Versions*.

6.6 Electrical Characteristics: 5 V

Specifications are for T_J = 25°C for the Figure 26 and Figure 32 (unless otherwise noted).

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
SYSTEM PARAMETERS TEST CIRCUIT Figure 26 and Figure 32(1)							
V_{OUT}	Output Voltage	V_{IN} = 12 V, I_{LOAD} = 0.5 A Circuit of Figure 26 and Figure 32		4.9	5	5.1	V
V_{OUT}	Output Voltage LM2576	0.5 A ≤ I_{LOAD} ≤ 3 A, 8 V ≤ V_{IN} ≤ 40 V Circuit of Figure 26 and Figure 32	T_J = 25°C	4.8	5	5.2	V
			Applies over full operating temperature range	4.75		5.25	
V_{OUT}	Output Voltage LM2576HV	0.5 A ≤ I_{LOAD} ≤ 3 A, 8 V ≤ V_{IN} ≤ 60 V Circuit of Figure 26 and Figure 32	T_J = 25°C	4.8	5	4.75	V
			Applies over full operating temperature range	5.225		5.275	
η	Efficiency	V_{IN} = 12 V, I_{LOAD} = 3 A			77%		

(1) External components such as the catch diode, inductor, input and output capacitors can affect switching regulator system performance. When the LM2576/LM2576HV is used as shown in Figure 26 and Figure 32, system performance is as shown in *Electrical Characteristics: All Output Voltage Versions*.

6.7 Electrical Characteristics: 12 V

Specifications are for T_J = 25°C (unless otherwise noted).

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
SYSTEM PARAMETERS TEST CIRCUIT Figure 26 and Figure 32(1)							
V_{OUT}	Output Voltage	V_{IN} = 25 V, I_{LOAD} = 0.5 A Circuit of Figure 26 and Figure 32		11.76	12	12.24	V
V_{OUT}	Output Voltage LM2576	0.5 A ≤ I_{LOAD} ≤ 3 A, 15 V ≤ V_{IN} ≤ 40 V Circuit of Figure 26 and Figure 32 and	T_J = 25°C	11.52	12	12.48	V
			Applies over full operating temperature range	11.4		12.6	
V_{OUT}	Output Voltage LM2576HV	0.5 A ≤ I_{LOAD} ≤ 3 A, 15 V ≤ V_{IN} ≤ 60 V Circuit of Figure 26 and Figure 32	T_J = 25°C	11.52	12	12.54	V
			Applies over full operating temperature range	11.4		12.66	
η	Efficiency	V_{IN} = 15 V, I_{LOAD} = 3 A			88%		

(1) External components such as the catch diode, inductor, input and output capacitors can affect switching regulator system performance. When the LM2576/LM2576HV is used as shown in Figure 26 and Figure 32, system performance is as shown in *Electrical Characteristics: All Output Voltage Versions*.

6.8 Electrical Characteristics: 15 V

over operating free-air temperature range (unless otherwise noted).

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
SYSTEM PARAMETERS TEST CIRCUIT Figure 26 and Figure 32(1)							
V_{OUT}	Output Voltage	V_{IN} = 25 V, I_{LOAD} = 0.5 A Circuit of Figure 26 and Figure 32		14.7	15	15.3	V
V_{OUT}	Output Voltage LM2576	0.5 A ≤ I_{LOAD} ≤ 3 A, 18 V ≤ V_{IN} ≤ 40 V Circuit of Figure 26 and Figure 32	T_J = 25°C	14.4	15	15.6	V
			Applies over full operating temperature range	14.25		15.75	
V_{OUT}	Output Voltage LM2576HV	0.5 A ≤ I_{LOAD} ≤ 3 A, 18 V ≤ V_{IN} ≤ 60 V Circuit of Figure 26 and Figure 32	T_J = 25°C	14.4	15	14.25	V
			Applies over full operating temperature range	15.68		15.83	
η	Efficiency	V_{IN} = 18 V, I_{LOAD} = 3 A			88%		

(1) External components such as the catch diode, inductor, input and output capacitors can affect switching regulator system performance. When the LM2576/LM2576HV is used as shown in Figure 26 and Figure 32, system performance is as shown in *Electrical Characteristics: All Output Voltage Versions*.

6.9 Electrical Characteristics: Adjustable Output Voltage

over operating free-air temperature range (unless otherwise noted).

PARAMETER		TEST CONDITIONS		MIN	TYP	MAX	UNIT
SYSTEM PARAMETERS TEST CIRCUIT Figure 26 and Figure 32(1)							
V_{OUT}	Feedback voltage	V_{IN} = 12 V, I_{LOAD} = 0.5 A V_{OUT} = 5 V, Circuit of Figure 26 and Figure 32		1.217	1.23	1.243	V
V_{OUT}	Feedback Voltage LM2576	0.5 A ≤ I_{LOAD} ≤ 3 A, 8 V ≤ V_{IN} ≤ 40 V V_{OUT} = 5 V, Circuit of Figure 26 and Figure 32	T_J = 25°C	1.193	1.23	1.267	V
			Applies over full operating temperature range	1.18		1.28	
V_{OUT}	Feedback Voltage LM2576HV	0.5 A ≤ I_{LOAD} ≤ 3 A, 8 V ≤ V_{IN} ≤ 60 V V_{OUT} = 5 V, Circuit of Figure 26 and Figure 32	T_J = 25°C	1.193	1.23	1.273	V
			Applies over full operating temperature range	1.18		1.286	
η	Efficiency	V_{IN} = 12 V, I_{LOAD} = 3 A, V_{OUT} = 5 V			77%		

(1) External components such as the catch diode, inductor, input and output capacitors can affect switching regulator system performance. When the LM2576/LM2576HV is used as shown in Figure 26 and Figure 32, system performance is as shown in *Electrical Characteristics: All Output Voltage Versions*.

6.10 Electrical Characteristics: All Output Voltage Versions

over operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP(1)	MAX	UNIT
SYSTEM PARAMETERS TEST CIRCUIT Figure 26 and Figure 32(2)							
I_b	Feedback Bias Current	V_{OUT} = 5 V (Adjustable Version Only)	T_J = 25°C	100	50		nA
			Applies over full operating temperature range	500			
f_O	Oscillator Frequency(3)	T_J = 25°C		47	52	58	kHz
		Applies over full operating temperature range		42		63	

(1) All limits specified at room temperature (25°C) unless otherwise noted. All room temperature limits are 100% production tested. All limits at temperature extremes are specified through correlation using standard Statistical Quality Control (SQC) methods.
(2) External components such as the catch diode, inductor, input and output capacitors can affect switching regulator system performance. When the LM2576/LM2576HV is used as shown in Figure 26 and Figure 32, system performance is as shown in *Electrical Characteristics: All Output Voltage Versions*.
(3) The oscillator frequency reduces to approximately 11 kHz in the event of an output short or an overload which causes the regulated output voltage to drop approximately 40% from the nominal output voltage. This self protection feature lowers the average power dissipation of the IC by lowering the minimum duty cycle from 5% down to approximately 2%.

Electrical Characteristics: All Output Voltage Versions (continued)

over operating free-air temperature range (unless otherwise noted)

PARAMETER		TEST CONDITIONS		MIN	TYP(1)	MAX	UNIT
V_{SAT}	Saturation Voltage	I_{OUT} = 3 A (4)	T_J = 25°C		1.4	1.8	V
			Applies over full operating temperature range			2	
DC	Max Duty Cycle (ON)(5)			93%	98%		
I_{CL}	Current Limit(4)(3)	T_J = 25°C		4.2	5.8	6.9	A
		Applies over full operating temperature range		3.5		7.5	
I_L	Output Leakage Current	Output = 0 V Output = −1 V Output = −1 V (6)(7)		2	7.5	30	mA
I_Q	Quiescent Current(6)				5	10	mA
I_{STBY}	Standby Quiescent Current	$\overline{ON}$ /OFF Pin = 5 V (OFF)			50	200	µA
$\overline{ON}$ /OFF CONTROL TEST CIRCUIT Figure 26 and Figure 32							
V_{IH}	$\overline{ON}$ /OFF Pin Logic Input Level	V_{OUT} = 0 V	T_J = 25°C	2.2	1.4		V
			Applies over full operating temperature range	2.4			
V_{IL}		V_{OUT} = Nominal Output Voltage	T_J = 25°C		1.2	1	V
			Applies over full operating temperature range			0.8	
I_{IH}	$\overline{ON}$ /OFF Pin Input Current	$\overline{ON}$ /OFF Pin = 5 V (OFF)			12	30	µA
I_{IL}		$\overline{ON}$ /OFF Pin = 0 V (ON)			0	10	µA

(4) Output pin sourcing current. No diode, inductor or capacitor connected to output.
(5) Feedback pin removed from output and connected to 0V.
(6) Feedback pin removed from output and connected to +12 V for the Adjustable, 3.3-V, and 5-V versions, and +25 V for the 12-V and 15-V versions, to force the output transistor OFF.
(7) V_{IN} = 40 V (60 V for high voltage version).

The following python scripts are run on the raspberry pi and handle communication between the smartphone application and flight controller.

Server.py – Main communication server that runs on the raspberry pi as the navigation controller.

```
import SocketServer
import sys
import os
import subprocess
import logging
import Config
import time
import shlex
from threading import Thread
from enum import Enum
from BinaryStream import BinaryStream
from BinaryStream import BinaryStreamEOFException
from dronekit import connect, VehicleMode
from pymavlink import mavutil
from Takeoff import TakeoffThread
from OpenSSL import SSL
import ConfigParser

logging.basicConfig(filename='Hovercam_Server.log', level=logging.INFO)
class Commands(Enum):
    LOGIN = 0
    TAKEOFF = 1
    HOVER = 2
    LAND = 3
    SET_DISTANCE = 4 # Get in future, instead of Set
    SET_GPS = 5 # Get in future, instead of Set
    SET_BATTERY_LIFE = 6 # Get in future, instead of Set
    SET_TARGET = 7
    REQUEST_VIDEO = 8

def startVideoStreamAndRecording():
    i = 0
    while os.path.exists("/home/pi/Desktop/Video/hovercam_%s_%s.h264" %
(time.strftime("%m-%d-%Y"), i)):
        i+= 1
    filename = "/home/pi/Desktop/Video/hovercam_%s_%s.h264" %
(time.strftime("%m-%d-%Y"), i)
    cmd = 'raspivid -vf -hf -t 0 -b 5000000 -w 854 -h 480 -fps 30 -o  - |
tee %s | gst-launch-1.0 -e -v fdsrc ! h264parse ! rtph264pay config-
interval=1 pt=96 ! gdppay ! tcpserversink host=192.168.10.1 port=5000' %
filename
    # other resolutions: 1280x720, 854x480
    global p
    p = subprocess.Popen(cmd, shell=True, stdout=subprocess.PIPE)

class SSlSocketServer(SocketServer.ThreadingMixIn, SocketServer.TCPServer):

    def __init__(self, server_address, RequestHandlerClass,
bind_and_activate=True):
        SocketServer.BaseServer.__init__(self, server_address,
```

```
            RequestHandlerClass)
        ctx = SSL.Context(SSL.SSLv3_METHOD)
        cert = 'client_cert.pem'
        key = 'private_key.pem'
        ctx.use_privatekey_file(key)
        ctx.use_certificate_file(cert)
        self.socket = SSL.Connection(ctx, socket.socket(self.address_family,
            self.socket_type))
        if bind_and_activate:
            self.server_bind()
            self.server_activate()
    def shutdown_request(self,request):
        request.shutdown()

class Decoder(SocketServer.StreamRequestHandler):
    """
    The RequestHandler/Decoder class for our server. This handles commands
sent from
    the smartphone application.
    """
    def setup(self):
        self.connection = self.request
        self.rfile = socket._fileobject(self.request, "rb", self.rbufsize)
        self.wfile = socket._fileobject(self.request, "wb", self.wbufsize)
    def parse(self, stream, vehicle):
        binaryStream = BinaryStream(stream)
        global thread
        thread = None
        flag = False
        try:
            packetId = binaryStream.read('int32')
            #print("packetID: ", packetId, Commands.HOVER)
            if packetId > -1 and packetId < 11: # Valid Packet Range
                if Commands(packetId) == Commands.LOGIN:
                    password = binaryStream.read('string')
                    config = ConfigParser.ConfigParser()
                    config.read("user.conf")
                    saltedHashPassword = config.get("myvars", "password")
                    if password == saltedHashPassword:
                        binaryStream.write('int32', 1) # true
                    else:
                        binaryStream.write('int32', 0) # false
                        return False
                elif Commands(packetId) == Commands.TAKEOFF:
                    print('Taking off...')
                    logging.info('Taking off...')
                    if thread is None:
                        thread = TakeoffThread(vehicle, binaryStream)
                        thread.start()
                    else:
                        if not thread.isAlive():
                            thread = TakeoffThread(vehicle, binaryStream)
                            thread.start()
                        else:
                            print("Taking off thread already exists.")
                            thead.join() # kill thread
                            i = 10 # try to start new thread within 10
```

```
seconds of killing old thread
                            while i < 10:
                                if not thread.isAlive():
                                    thread = TakeoffThread(vehicle)
                                    thread.start()
                                    flag = True
                                    break
                                i += 1
                                time.sleep(1)
                    elif Commands(packetId) == Commands.HOVER:
                        #hover
                        print('Hovering...')
                        logging.info('Hovering')
                        if thread is not None:
                            if thread.isAlive():
                                thread.join()
                            else:
                                print("Thread is dead")
                        vehicle.mode = VehicleMode("LOITER")
                    elif Commands(packetId) == Commands.LAND:
                        #Land
                        print('Landing...')
                        logging.info('Landing...')
                        if thread is not None:
                            if thread.isAlive():
                                thread.join()
                            else:
                                print("Thread is dead")
                        vehicle.mode = VehicleMode("LAND")
                    elif Commands(packetId) == Commands.SET_DISTANCE:
                        #Set Distance
                        distance = binaryStream.read('int32')
                        print('Setting distance to ', distance, 'ft')
                        logging.info('Setting distance to: ' + str(distance) +
'ft')
                        Config.distance = distance
                    elif Commands(packetId) == Commands.SET_GPS:
                        #SET_GPS
                        print('Reading and Setting GPS...')
                        logging.info('Reading and setting GPS')
                        Config.longitude =
binaryStream.read('string').decode('UTF-8')
                        Config.latitude =
binaryStream.read('string').decode('UTF-8')
                else:
                    logging.info('Invalid packetId: ', packetId)
            except BinaryStreamEOFException:
                # One of our attempts to read a field went beyond the end of the
file.
                print("Error: Connection has been terminated.")
                logging.error('Error: Connection has been terminated')
                print("Landing drone...")
                if thread is not None:
                    if thread.isAlive():
                        thread.join()
                    else:
                        print("Thread is dead")
```

```
            logging.info('Landing drone after disconnection.')
            vehicle.mode = VehicleMode("RTL") # Return to home location and
land if disconnected from client
            p.kill()
            return False
        return True;
    def handle(self):
        flag = True;
        print("Establishing new connection")
        logging.info('Connection to client established')
        vehicle = connect('/dev/ttyS0', wait_ready=True, baud=57600)
        logging.info('Connected to vehicle')
        print("Connected to vehicle")
        request = self.connection
        try: # Update the smartphone application to allow for commands to be
sent
            binaryStream = BinaryStream(request)
            binaryStream.write('int32', 0)
            binaryStream.write('string', "Vehicle is connected")
            binaryStream.write('int32', 2)
            binaryStream.write('string', "Ready for commands.")
        except:
            print("Error sending messages")
        print("Connection messages sent.")
        vehicle.mode = VehicleMode("GUIDED")
        while flag:
            flag = self.parse(request, vehicle)

if __name__ == "__main__":
    HOST, PORT = "0.0.0.0", 43594 # for linux, use 0.0.0.0
    startVideoStreamAndRecording()
    server = SSlSocketServer((HOST, PORT), Decoder)
    logging.info('Starting Hovercam')
    # The server will keep running until it is interrupted by pressing Ctrl-
C
    server.serve_forever()
```

Takeoff.py – Script that handles moving the quadcopter in takeoff mode

```
import io
import socket
import struct
import time
import math
import geopy
import threading
import Config
import logging
from geopy.distance import VincentyDistance
from dronekit import VehicleMode, LocationGlobal, LocationGlobalRelative,
connect
from pymavlink import mavutil
```

```
from time import sleep
from ObstacleAvoidance import startObstacleAvoidance

logging.basicConfig(filename='Hovercam_Takeoff.log', level=logging.INFO)
class TakeoffThread(threading.Thread):
    ##def CalculateDistance(lat1, long1, lat2, long2):
    ##    distance = gpxpy.geo.haversine_distance(lat1, long1, lat2, long2)
    ##    return distance
    ##def CalculateDirection(lat1, long1, lat2, long2):
    ##    earing = atan2(sin(long2-long1)*cos(lat2), cos(lat1)*sin(lat2)-
sin(lat1)*cos(lat2)*cos(long2-long1))
    ##    bearing = degrees(bearing)
    ##    bearing = (bearing + 360) % 360
    ##    return bearing

    def get_distance_meters(self, location1, location2):
        lat = location2.lat - location1.lat
        long = location2.lon - location1.lon
        return math.sqrt((lat*lat)+(long*long)) * 1.113195e5
    def getReducedPosition(self, latitude, longitude, bearing, distance):
        origin = geopy.Point(latitude, longitude)
        d = distance / 1000 # mto km
        destination = VincentyDistance(kilometers=d).destination(origin,
bearing)
        return destination.latitude, destination.longitude
    def getBearing(self, lat1, long1, lat2, long2):
        off_x = long2 - long1
        off_y = lat2 - lat1
        bearing = 90.00 + math.atan2(-off_y, off_x) * 57.2957795
        if bearing < 0:
            bearing += 360.00
        return bearing
    def sendMessage(self, i, message):
        try:
            self.stream.write('int32', i)
            self.stream.write('string', message)
        except:
            print "Connection was dropped."
    def gotoAltitude(self):
        aTargetAltitude = Config.altitude
        print("Target altitude: " + str(aTargetAltitude) + ", current alt: "
+ str(self.vehicle.location.global_relative_frame.alt))
        if self.vehicle.location.global_relative_frame.alt > aTargetAltitude
* 1.05 or self.vehicle.location.global_relative_frame.alt < aTargetAltitude
* 0.95:
            print("current altitude: " +
str(self.vehicle.location.global_relative_frame.alt))
            self.vehicle.simple_takeoff(aTargetAltitude) # Take off to
target altitude
            #Wait until the vehicle reaches a safe height before processing
the goto (otherwise the command
            #after Vehicle.simple_takeoff will execute immediately).
            #time.sleep(3)
            i2 = 0
            while True and (i2 <= 3) and not self._stopevent.isSet():
                print " Altitude: ",
self.vehicle.location.global_relative_frame.alt
```

```
                if self.vehicle.location.global_relative_frame.alt <=
aTargetAltitude*1.05 and self.vehicle.location.global_relative_frame.alt >=
aTargetAltitude*0.95: #Trigger just below target alt.
                    print "Reached target altitude"
                    break
                i2 += 1
                time.sleep(1)
    def arm_and_takeoff(self, aTargetAltitude):
        """
        Arms vehicle and fly to aTargetAltitude.
        """
        print "Basic pre-arm checks"
        self.sendMessage(2, "Running basic pre-arm checks..")
        # Don't let the user try to arm until autopilot is
        i = 0
        while not self.vehicle.is_armable and (i != 15) and not
self._stopevent.isSet():
             print " Waiting for vehicle to initialize..."
             logging.info("Waiting for vehicle to initialize...")
             print "gps: ", self.vehicle.gps_0.fix_type
             time.sleep(1)
             i += 1
        if i == 15:
            print("Microcontroller could not initialize.")
            logging.info("Microcontroller could not initialize.")
            self.sendMessage(1, "Microcontroller could not initialize.")
            self._stopevent.set()
            if self.thread is not None:
                if self.thread.isAlive():
                    self.thread.join()
            return False

        print "Arming motors"
        self.sendMessage(2, "Arming motors...")
        # Copter should arm in GUIDED mode
        self.vehicle.mode = VehicleMode("GUIDED")
        self.vehicle.armed = True
        i = 0
        while not self.vehicle.armed and (i != 15)  and not
self._stopevent.isSet():
            print " Waiting for arming..."
            logging.info('Waiting for arming...')
            time.sleep(1)
            i += 1
            #if i == 15:
              #  print("Quadcopter stuck in arming loop. Reconnecting
drone...")
                #self.vehicle.close();
                #self.vehicle = connect('/dev/ttyS0', wait_ready=True,
baud=57600)
        if i == 15:
            print ("Quadcopter stuck in arming loop.")
            logging.info("Quadcopter stuck in arming loop.")
            self.sendMessage(1, "Microcontroller could not arm motors.")
            self._stopevent.set()
            if self.thread is not None:
                if self.thread.isAlive():
```

```
                    self.thread.join()
            return False
        print "Taking off!"
        self.sendMessage(2, "Vehicle is taking off.")
        self.vehicle.simple_takeoff(aTargetAltitude) # Take off to target
altitude

         #Wait until the vehicle reaches a safe height before processing the
goto (otherwise the command
          #after Vehicle.simple_takeoff will execute immediately).
        time.sleep(5)
        i2 = 0
        while True and (i2 <= 15) and not self._stopevent.isSet():
            print " Altitude: ",
self.vehicle.location.global_relative_frame.alt
            if
self.vehicle.location.global_relative_frame.alt>=aTargetAltitude*0.95:
#Trigger just below target alt.
                print "Reached target altitude"
                break
            i2 += 1
            time.sleep(1)
        self.sendMessage(2, "Altitude has been adjusted to set height.")
        return True
    def __init__(self,vehicle, stream, name='TakeoffThread'):
        self._stopevent = threading.Event()
        self._sleepperiod = 1.0
        self.vehicle = vehicle
        self.stream = stream
        self.thread = None
        threading.Thread.__init__(self, name=name)
    def getNewPosition(self, target, pos):
        if (pos - target) > 0:
            target -= 2
        else:
            target += 2
    def gotoGPS(self, location):
        currentLocation = self.vehicle.location.global_frame
        distance = self.get_distance_meters(currentLocation, location)
        print "Distance to target: " + str(distance)
        if distance < 50 and distance > Config.distance * 0.3048:
            self.vehicle.simple_goto(location)
##            while not self._stopevent.isSet() and self.vehicle.mode.name==
"GUIDED":
##                distanceToTarget =
self.get_distance_meters(self.vehicle.location.global_frame, location)
##                print "distance to target: " + str(distanceToTarget)
##                if distanceToTarget < distance*0.95: # add little
threshold to meet target
##                    print "GPS target reached"
##                    logging.info("GPS target reached")
##                    break;
##            time.sleep(1)
        else:
            print "Distance to GPS target is too far or close: "  +
str(distance)
            logging.info("Distnace to GPS target is too far or close: " +
```

```
str(distance))
    def run(self):
        print "Takeoff thread started."
        takeoffSuccess = self.arm_and_takeoff(Config.altitude)
        # Commented out obstacle avoidance:
        #self.startObstacleAvoidance()
        #self.thread = startObstacleAvoidance(self.vehicle)
        #self.thread.start()
        if takeoffSuccess:
            self.sendMessage(2, "Following target..")
            while not self._stopevent.isSet() and self.vehicle.mode.name ==
"GUIDED":
                # If Image processing
                # else, gps
                #print "Sending location: ", Config.latitude,
Config.longitude
                srcLatitude = self.vehicle.location.global_frame.lat
                srcLongitude = self.vehicle.location.global_frame.lon
                dstLatitude = float(Config.latitude)
                dstLongitude = float(Config.longitude)
                print "source      lat: " + str(srcLatitude) + " long: " +
str(srcLongitude)
                print "destination lat: " + str(dstLatitude) + " long: " +
str(dstLongitude)
                logging.info("source lat: " + str(srcLatitude) + " long: " +
str(srcLongitude))
                logging.info("destination lat: " + str(dstLatitude) + "
long: " + str(dstLongitude))
                #distance = Config.distance * 0.3048
                #target = LocationGlobal(dstLatitude, dstLongitude)
    ##            if False:
#self.get_distance_meters(self.vehicle.location.global_frame, target) >
distance:
    ##                bearing = self.getBearing(srcLatitude, srcLongitude,
dstLatitude, dstLongitude)
    ##                lat, long = self.getReducedPosition(dstLatitude,
dstLongitude, distance, bearing)
    ##                print "new lat: ", lat, " long: ", long
    ##                logging.info("new lat: " +str(lat) + " long: "+
str(long))
    ##                location = LocationGlobalRelative(lat, long,
Config.altitude)
                target = LocationGlobalRelative(dstLatitude, dstLongitude,
Config.altitude)
                self.gotoGPS(target) #,second parameter: groundspeed=10
                time.sleep(2) # wait 5 seconds before setting GPS
                self.gotoAltitude()
            self.sendMessage(2, "Following has ended...")
        print "Takeoff thread ending.."

        # Commented out obstacle avoidance:
##        if self.thread is not None:
##            if self.thread.isAlive():
##                self.thread.join()
##            else:
##                print "Obstacle avoidance thread is dead."
##        else:
```

```
##            print "Obstacle avoidance thread does not exist."
    def join(self, timeout=None):
        # Stop thread
        self._stopevent.set()
        if self.thread is not None:
            if self.thread.isAlive():
                self.thread.join()
        threading.Thread.join(self, timeout)
```

Config.py – Script that contains the configuration variables

```
latitude = 0
longitude = 0
altitude = 4
distance = 5
isObstacleDeteched = False
isLeft = False
isRight = False
```

BinaryStream.py – Script that contains methods to write data types to a TCP stream, especially to a c# TCP Stream

```
import struct
import sys

class BinaryStreamEOFException(Exception):
    def __init__(self):
        pass
    def __str__(self):
        return 'Not enough bytes in file to satisfy read request'
"""
 This class is used to write data types to the connection stream. This is
set to the format that will allow
 the c# Tcp Client libraries to interpret.
"""
class BinaryStream:
    # Map well-known type names into struct format characters.
    typeNames = {
        'int8'   :'b',
        'uint8'  :'B',
        'int16'  :'h',
        'uint16' :'H',
        'int32'  :'i',
        'uint32' :'I',
        'int64'  :'q',
        'uint64' :'Q',
        'float'  :'f',
        'double' :'d',
        'char'   :'s',
        'string' :'string'} # add special case string if it is required

    def __init__(self, stream):
        self.stream = stream
```

```
    def byte_length(self, i):
        return (i.bit_length() + 7) // 8
    def utf8len(self, s):
        return len(s.encode('utf-8'))
    def read(self, typeName):
        typeFormat = BinaryStream.typeNames[typeName.lower()]
        typeSize = 0
        if typeFormat == 'string':
            typeSize = self.read('uint8')
            value = self.stream.recv(typeSize)
            return value
        else:
            typeSize = struct.calcsize(typeFormat)
            value = self.stream.recv(typeSize)
        if typeSize != len(value):
            raise BinaryStreamEOFException
        unpacked_bytes = struct.unpack(typeFormat, value)
        return unpacked_bytes[0]

    def write(self, typeName, value):
        typeFormat = BinaryStream.typeNames[typeName.lower()]
        if typeFormat == 'string':
            length = self.utf8len(value)
            self.write('uint8', length)
            self.stream.send(value.encode())
        else:
            typeSize = struct.calcsize(typeFormat)
                  # Removed value comparison, as we sometimes cast values
            #value = self.stream.recv(typeSize)
            #print("TypeSize: ", typeSize)
            #print("actual size: " , self.byte_length(value))
            #if typeSize != self.byte_length(value):
              #  raise BinaryStreamEOFException
            packed_bytes = struct.pack(typeFormat, value)
            self.stream.send(packed_bytes)
```

The following classes consist of the smartphone application written in c# using Xamarin in Visual Studio:

Login.cs – Handles the Login screen activity

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using System.Net.Security;
using System.IO;
```

```
namespace HovercamMobile
{
    [Activity(Label = "Login")]
    public class Login : Activity
    {
        private EditText password;
        private Button login;
        private ProgressBar ConnectionProgress;

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            Window.RequestFeature(Android.Views.WindowFeatures.NoTitle);

            SetContentView(Resource.Layout.Login_Screen);

            //Login variables
            login = FindViewById<Button>(Resource.Id.Login);
            password = FindViewById<EditText>(Resource.Id.Login_Box);
            ConnectionProgress =
FindViewById<ProgressBar>(Resource.Id.ConnectionProgress);
            login.Click += login_Click;

        }

        void login_Click(object sender, EventArgs e)
        {
            ConnectionProgress.Visibility = ViewStates.Visible;

            //Login process
            // bool areEqual = String.Equals(sPassword, password.Text,
StringComparison.Ordinal);


            try
            {
                SslStream client = ClientSocket.Instance;
                BinaryWriter writer = new BinaryWriter(client, Encoding.UTF8, true);
                writer.Write((int)Commands.LOGIN);
                writer.Write(password.Text);
                writer.Close();
                BinaryReader reader = new BinaryReader(client, Encoding.UTF8, true);
                int isValid = reader.ReadInt32();
                if (isValid == 0)
                {

                    new AlertDialog.Builder(this).SetPositiveButton("Close", (s, args)
=>
                    {
                            // User pressed yes
                        }).SetMessage("Incorrect password
entered!").SetTitle("Error").Show();
                }
                else
                {
                    Finish();
```

```
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine(ex);
                ConnectionProgress.Visibility = ViewStates.Invisible;
            }


        }
    }
}
```

MainActivity.cs – Handles the main window activity with the control buttons

```
using Android.App;
using Android.Widget;
using Android.OS;
using System;
using System.Threading;
using Android.Content;
using Android.Locations;
using System.Collections.Generic;
using Android.Util;
using System.Linq;
using System.Text;
using Android.Runtime;
using System.Net.Security;
using System.IO;

namespace HovercamMobile
{
    [Activity(Label = "HovercamMobile", MainLauncher = true, Icon =
"@drawable/Quadcopter")]
    public class MainActivity : Activity, ILocationListener
    {
        //Global variables
        System.Timers.Timer timer;
        private TextView Textview1;
        /*LocationManager locMgr;
        Location currentLocation;*/
        public static Location _currentLocation;
        public static Location _previousLocation;
        LocationManager _locationManager;
        String _locationProvider;
        TextView latitude;
        TextView longitude;
        //string tag = "MainActivity";

        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            Window.RequestFeature(Android.Views.WindowFeatures.NoTitle);
            StartActivity(typeof(Login));
```

```
            //Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);

            Button Set_Distance = FindViewById<Button>(Resource.Id.Set_Distance);
            TextView Textview1 = FindViewById<TextView>(Resource.Id.Yo);
            Button Take_Off = FindViewById<Button>(Resource.Id.Take_Off);
            Button Hover = FindViewById<Button>(Resource.Id.Hover);
            Button Land = FindViewById<Button>(Resource.Id.Land);
            Button Video_Stream = FindViewById<Button>(Resource.Id.VideoStream);


            //Clicking Set Distance button sends you to Set Distance Screen
            Set_Distance.Click += delegate
            {
                StartActivity(typeof(Set_Distance_Activity));
            };

            //Clicking Video Stream button sends you to Video Stream Screen
            Video_Stream.Click += delegate
            {
                StartActivity(typeof(VideoStream));
            };

            //Changes textview1 based on button clicked and a 3 sec timer
            Take_Off.Click += delegate
            {
                //Textview1.Text = "Take Off sequence initiated...";
                timer = new System.Timers.Timer();
                timer.Enabled = true;
                timer.Interval = 3000; //3 seconds
                timer.Elapsed += new System.Timers.ElapsedEventHandler(t_Elapsed);
            };


            //Take Off, Hover, and Landing Sequences

            Take_Off.Click += delegate
            {
                Take_Off.Enabled = false;
                Hover.Enabled = true;
                Land.Enabled = true;
                try
                {
                    SslStream client = ClientSocket.Instance;
                    BinaryWriter writer = new BinaryWriter(client, Encoding.UTF8,
true);
                    writer.Write((int)Commands.TAKEOFF);
                    writer.Close();
                }
                catch (Exception ex)
                {
                    new AlertDialog.Builder(this).SetPositiveButton("Close", (sender,
args) =>
                    {
                        // User pressed yes
                    }).SetMessage("The connection to the raspberry pi has been lost!\n"
+ ex.Message).SetTitle("Error").Show();
                    Take_Off.Enabled = false;
```

```
                    Hover.Enabled = true;
                }
            };
        protected override void OnCreate(Bundle bundle)
        {
            base.OnCreate(bundle);

            Window.RequestFeature(Android.Views.WindowFeatures.NoTitle);
            StartActivity(typeof(Login));

            //Set our view from the "main" layout resource
            SetContentView(Resource.Layout.Main);

            Button Set_Distance = FindViewById<Button>(Resource.Id.Set_Distance);
            TextView Textview1 = FindViewById<TextView>(Resource.Id.Yo);
            Button Take_Off = FindViewById<Button>(Resource.Id.Take_Off);
            Button Hover = FindViewById<Button>(Resource.Id.Hover);
            Button Land = FindViewById<Button>(Resource.Id.Land);
            Button Video_Stream = FindViewById<Button>(Resource.Id.VideoStream);


            //Clicking Set Distance button sends you to Set Distance Screen
            Set_Distance.Click += delegate
            {
                StartActivity(typeof(Set_Distance_Activity));
            };

            //Clicking Video Stream button sends you to Video Stream Screen
            Video_Stream.Click += delegate
            {
                StartActivity(typeof(VideoStream));
            };

            //Changes textview1 based on button clicked and a 3 sec timer
            Take_Off.Click += delegate
            {
                //Textview1.Text = "Take Off sequence initiated...";
                timer = new System.Timers.Timer();
                timer.Enabled = true;
                timer.Interval = 3000; //3 seconds
                timer.Elapsed += new System.Timers.ElapsedEventHandler(t_Elapsed);
            };


            //Take Off, Hover, and Landing Sequences

            Take_Off.Click += delegate
            {
                Take_Off.Enabled = false;
                Hover.Enabled = true;
                Land.Enabled = true;
                try
                {
                    SslStream client = ClientSocket.Instance;
                    BinaryWriter writer = new BinaryWriter(client, Encoding.UTF8,
true);
                    writer.Write((int)Commands.TAKEOFF);
                    writer.Close();
```

```
                }
                catch (Exception ex)
                {
                    new AlertDialog.Builder(this).SetPositiveButton("Close", (sender,
args) =>
                    {
                        // User pressed yes
                    }).SetMessage("The connection to the raspberry pi has been lost!\n"
+ ex.Message).SetTitle("Error").Show();
                    Take_Off.Enabled = false;
                    Hover.Enabled = true;
                }
            };

            Hover.Click += delegate
            {
                Hover.Enabled = false;
                Land.Enabled = true;
                Take_Off.Enabled = true;
                try
                {
                    SslStream client = ClientSocket.Instance;
                    BinaryWriter writer = new BinaryWriter(client, Encoding.UTF8,
true);
                    writer.Write((int)Commands.HOVER);
                    writer.Close();
                }
                catch (Exception ex)
                {
                    new AlertDialog.Builder(this).SetPositiveButton("Close", (sender,
args) =>
                    {
                        // User pressed yes
                    }).SetMessage("The connection to the raspberry pi has been lost!\n"
+ ex.Message).SetTitle("Error").Show();
                }
            };

            Land.Click += delegate
            {
                Take_Off.Enabled = true;
                Land.Enabled = false;
                Hover.Enabled = false;
                try
                {
                    SslStream client = ClientSocket.Instance;
                    BinaryWriter writer = new BinaryWriter(client, Encoding.UTF8,
true);
                    writer.Write((int)Commands.LAND);
                    writer.Close();
                }
                catch (Exception ex)
                {
                    new AlertDialog.Builder(this).SetPositiveButton("Close", (sender,
args) =>
                    {
                        // User pressed yes
```

```
                    }).SetMessage("The connection to the raspberry pi has been lost!\n"
+ ex.Message).SetTitle("Error").Show();
                }
            };


            //Shows dialog box with progress bar. Not actually goning to be used but i
may want to reference it for something later
            //Land.Click += BatteryLife_Indicator;

            //Thread for progressBar displaying battery consumption
         /*   new Thread(new ThreadStart(delegate
            {
                int progressBarStatus = 100;
                BatteryLife.Progress = 98;

                //while (true)
                //{
                while (progressBarStatus > 5)
                {
                    progressBarStatus -= 5;
                    BatteryLife.Progress = progressBarStatus;
                    Thread.Sleep(60000); // Sleep 1000ms
                }

                this.RunOnUiThread(() => new
AlertDialog.Builder(this).SetPositiveButton("Close", (sender, args) =>
                {
                    // User pressed yes
                }).SetMessage("Battery is low!").SetTitle("Warning").Show());
                //BatteryLife.Progress = 100;
                //progressBarStatus = 100;
                //}
            })).Start();*/

            // receive data
            new Thread(() => // Thread (like Timer)
            {
                try
                {
                    SslStream client = ClientSocket.Instance;
                    BinaryReader reader = null;
                    string message = string.Empty;
                    int messageId = -1;
                    while (true)
                    {
                        reader = new BinaryReader(client, Encoding.UTF8, true);
                        /*
                         *  0 - Vehicle is connected
                         *  1 - Error message
                         *  2 - Vehicle state
                         * */
                        messageId = reader.ReadInt32();
                        message = reader.ReadString();
                        if (messageId == 0)
                        {
                            this.RunOnUiThread(() =>
                            {
```

```
                                Set_Distance.Enabled = true;
                                Take_Off.Enabled = true;
                                Hover.Enabled = false;
                                Land.Enabled = false;
                            });
                        }
                        else if (messageId == 1)
                        {
                            this.RunOnUiThread(() => new
AlertDialog.Builder(this).SetPositiveButton("Close", (sender, args) =>
                                {
                                    // User pressed yes
                                }).SetMessage(message).SetTitle("Warning").Show());
                        }
                        else if (messageId == 2)
                        {

                            this.RunOnUiThread(() => { Textview1.Text = message; if
(message.Equals("Following has ended.")) { Take_Off.Enabled = true; } });
                        }
                        reader.Close();

                    }
                    //writer.Write((int)Commands.TAKEOFF);
                }
                catch (Exception ex)
                {
                    this.RunOnUiThread(() => new
AlertDialog.Builder(this).SetPositiveButton("Close", (sender, args) =>
                    {
                        // User pressed yes
                    }).SetMessage("The connection to the raspberry pi has been lost!\n"
+ ex.Message).SetTitle("Error").Show());
                }

            }).Start(); // Start the Thread
            //GPS

            latitude = FindViewById<TextView>(Resource.Id.latitude);
            longitude = FindViewById<TextView>(Resource.Id.longitude);

            // initialize location manager
            InitializeLocationManager();

        }
        public void InitializeLocationManager()
        {
            _locationManager = (LocationManager)GetSystemService(LocationService);
            Criteria criteriaForLocationService = new Criteria
            {
                Accuracy = Accuracy.Fine
            };
            IList<string> acceptableLocationProviders =
_locationManager.GetProviders(criteriaForLocationService, true);

            if (acceptableLocationProviders.Any())
            {
                _locationProvider = acceptableLocationProviders.First();
```

```
            }
            else
            {
                _locationProvider = String.Empty;
            }
        }
        public void OnLocationChanged(Location location)
        {
            try
            {
                Location loc = location;
                latitude.Text = "Latitude: " + loc.Latitude.ToString();
                longitude.Text = "Longitude: " + loc.Longitude.ToString();
                try
                {
                    SslStream client = ClientSocket.Instance;
                    BinaryWriter writer = new BinaryWriter(client, Encoding.UTF8,
true);
                    writer.Write((int)Commands.SET_GPS);
                    writer.Write(loc.Longitude.ToString());
                    writer.Write(loc.Latitude.ToString());
                    writer.Close();
                }
                catch (Exception ex)
                {
                    Console.WriteLine("Error sending GPS position: " + ex);
                    //new AlertDialog.Builder(this).SetPositiveButton("Okay", (sender,
args) =>
                    //{
                    //    // User pressed yes
                    //}).SetMessage("The connection to the raspberry pi has been
lost!\n" + ex.Message).SetTitle("Error").Show();
                }
                _currentLocation = loc;
                //longitude.Text = "Longitude: " + location.Longitude.ToString();

            }
            catch (Exception ex)
            {
                Console.WriteLine("error: " + ex);
                new AlertDialog.Builder(this).SetPositiveButton("Close", (sender, args)
=>
                {
                    // User pressed yes
                }).SetMessage("Error: " + ex.Message).SetTitle("Error").Show();
            }
        }

        public void OnProviderDisabled(string provider)
        {
        }

        public void OnProviderEnabled(string provider)
        {
            Console.WriteLine("Provider is enabled");
        }
```

```
        public void OnStatusChanged(string provider, Availability status, Bundle
extras)
        {
            Console.WriteLine(status.ToString());
        }

        protected override void OnResume()
        {
            base.OnResume();
            _locationManager.RequestLocationUpdates(_locationProvider, 1000, 0.5f,
this);
        }
        protected override void OnPause()
        {
            base.OnPause();
            //_locationManager.RemoveUpdates(this);
        }

        //Timer to change textview1
        private void t_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
        {
            RunOnUiThread(() =>
            {
                Textview1 = FindViewById<TextView>(Resource.Id.Yo);
                //Textview1.Text = "Take Off sequence complete.";
                timer.Stop();
            });
        }


    }
}
```

ClientSocket.cs – Handles the SSLStream connection as a singleton to allow for different activities to access the connection stream

```
using System;
using System.Net.Sockets;
using System.Security.Cryptography;
using System.Security.Cryptography.X509Certificates;
using System.Security.Authentication;
using System.Net.Security;

public class ClientSocket
{
    private static SslStream instance;
    private static TcpClient client;
    //Server certificate name
    private static readonly string ServerCertificateName = "MyServer";
    //Directory of client certificate used in SSL authentication
    private static readonly string ClientCertificateFile = "./Cert/client.pfx";
    private static readonly string ClientCertificatePassword = null;

    private ClientSocket() { }

    public static SslStream Instance
```

```
    {
        get
        {
            if (instance == null || !client.Connected)
            {
                client = new TcpClient();
                client.Connect("192.168.10.1", 43594);
                var clientCertificate = new X509Certificate2(ClientCertificateFile,
ClientCertificatePassword);
                var clientCertificateCollection = new X509CertificateCollection(new
X509Certificate[] { clientCertificate });
                instance = new SslStream(client.GetStream(), false,
App_CertificateValidation);
                Console.WriteLine("Client connected.");
                instance.AuthenticateAsClient(ServerCertificateName,
clientCertificateCollection, SslProtocols.Tls12, false);
            }
            return instance;
        }
    }
    /// <summary>
    /// Authenticates the client's certificate.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="certificate"></param>
    /// <param name="chain"></param>
    /// <param name="sslPolicyErrors"></param>
    /// <returns></returns>
    private static bool App_CertificateValidation(Object sender, X509Certificate
certificate, X509Chain chain, SslPolicyErrors sslPolicyErrors)
    {
        if (sslPolicyErrors == SslPolicyErrors.None) { return true; }
        if (sslPolicyErrors == SslPolicyErrors.RemoteCertificateChainErrors) { return
true; } //we don't have a proper certificate tree
        Console.WriteLine("*** SSL Error: " + sslPolicyErrors.ToString());
        return false;
    }
}
```

Commands.cs – Contains the enum for all the different commands

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;

namespace HovercamMobile
{
    public enum Commands
```

```
    {
        LOGIN = 0,
        TAKEOFF = 1,
        HOVER = 2,
        LAND = 3,
        SET_DISTANCE = 4,
        SET_GPS = 5,
        SET_BATTERY_LIFE = 6,
        SET_TARGET = 7,
        REQUEST_VIDEO = 8,
    }
}
```

Set_Distance.cs – The Set distance activity class to handle setting the distance

```
using Android.App;
using Android.Content;
using Android.OS;
using Android.Widget;
using Java.Lang;
using static Android.Widget.SeekBar;
using Android.Preferences;
using Android.Util;
using System.Net.Security;
using System.IO;
using System.Text;

namespace HovercamMobile
{
    [Activity(Label = "Set_Distance")]
    public class Set_Distance_Activity : Activity, SeekBar.IOnSeekBarChangeListener
    {
        SeekBar Set_Distance;
        TextView Display_Distance;
        Button Set_Button;
        int distance;
        protected override void OnDestroy()
        {
            base.OnDestroy();

            //Sets up preferences manager in order to save strings, ints, or bools on
closure of activity
            ISharedPreferences prefs =
PreferenceManager.GetDefaultSharedPreferences(Application.Context);
            ISharedPreferencesEditor editor = prefs.Edit();
            editor.PutInt("User_Distance_Set", Set_Distance.Progress);
            editor.Apply();        // applies changes asynchronously
        }

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);
            Window.RequestFeature(Android.Views.WindowFeatures.NoTitle);
            distance = 0;
            SetContentView(Resource.Layout.Set_Distance_Screen);
```

```
            // Seek Bar code for setting the distance
            Set_Distance = FindViewById<SeekBar>(Resource.Id.Set_Distance_SeekBar);
            Display_Distance = FindViewById<TextView>(Resource.Id.Display_Distance);

            //Resets the distance set to whatever it was when activity was destroyed
last
            ISharedPreferences prefs =
PreferenceManager.GetDefaultSharedPreferences(Application.Context);
            Set_Distance.Progress = prefs.GetInt("User_Distance_Set", 0);

            //Displays text displaying Distance Set
            Display_Distance.Text = string.Format("The user adjusted the value of the
SeekBar to {0}ft.", Set_Distance.Progress);

            Set_Distance.SetOnSeekBarChangeListener(this);
            //Clicking Back button sends you to the Main Screen
            Set_Button = FindViewById<Button>(Resource.Id.SDBack);
            Set_Button.Click += delegate
            {
                try
                {
                    SslStream client = ClientSocket.Instance;
                    BinaryWriter writer = new BinaryWriter(client, Encoding.UTF8,
true);
                    writer.Write((int)Commands.SET_DISTANCE);
                    writer.Write(distance);
                    writer.Close();
                    Finish();
                }
                catch (Exception ex)
                {
                    new AlertDialog.Builder(this).SetPositiveButton("Close", (sender,
args) =>
                    {
                        Finish();
                    }).SetMessage("The connection to the raspberry pi has been lost!\n"
+ ex.Message).SetTitle("Error").Show();
                }
            };

        }
        public void OnProgressChanged(SeekBar seekBar, int Progress, bool fromUser)
        {

            if (fromUser)
            {
                Display_Distance.Text = string.Format("The user adjusted the value of
the SeekBar to {0}ft.", seekBar.Progress + 1);
            }
            distance = seekBar.Progress;
        }

        public void OnStartTrackingTouch(SeekBar seekBar)
        {
            //System.Diagnostics.Debug.WriteLine("Tracking changes.");
        }

        public void OnStopTrackingTouch(SeekBar seekBar)
```

```
        {
            //System.Diagnostics.Debug.WriteLine("Stopped tracking changes.");
        }



    }
}
```

VideoStream.cs – Class that handles the backend of showing the TCP video stream from gstreamer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using Android.App;
using Android.Content;
using Android.OS;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using System.Net.Sockets;
using System.IO;
using Android.Webkit;
using Java.Lang;

namespace HovercamMobile
{
    [Activity(Label = "VideoStream", ScreenOrientation =
Android.Content.PM.ScreenOrientation.Landscape)]
    public class VideoStream : Activity
    {

        protected override void OnCreate(Bundle savedInstanceState)
        {
            base.OnCreate(savedInstanceState);

            Window.RequestFeature(Android.Views.WindowFeatures.NoTitle);

            SetContentView(Resource.Layout.Videostream_Screen);

            // Set web view content to be the raspberry pi camera stream
            /*   WebView webView = FindViewById<WebView>(Resource.Id.webView1);
                webView.Settings.JavaScriptEnabled = true;

                // Get the width and height of the view because its different for
different phone or table layouts
                // Pass these values to the URL in the web view to display the HTTP
stream
                int width = 640;
                int height = 320;
                string stream = "http://192.168.10.1:8080/stream";
                webView.LoadUrl(stream + "?width=" + width + "&height=" + height);
                */
            //Clicking Back button sends you to the Main Screen
            Intent intent = new Intent("pl.effisoft.rpicamviewer2.PREVIEW");
```

```
            int camera = 0;

            //--------- Basic settings
            intent.PutExtra("full_screen", true);

            intent.PutExtra("name" + camera, "My pipeline name");
            intent.PutExtra("host" + camera, "192.168.10.1");
            intent.PutExtra("port" + camera, 5000);
            intent.PutExtra("description" + camera, "My pipeline description");
            intent.PutExtra("uuid" + camera, System.Guid.NewGuid().ToString());
            intent.PutExtra("aspectRatio" + camera, 1.6);
            intent.PutExtra("autoplay" + camera, true);

            //--------- Enable advanced mode
            intent.PutExtra("advanced" + camera, true); //when advanced=true, then
custom_pipeline will be played
                                                        //when advanced=false, then
pipeline will be generated from host, port (use only for backward compatibility with
previous versions)
            intent.PutExtra("custom_pipeline" + camera, "tcpclientsrc host=192.168.10.1
port=5000 ! gdpdepay ! rtph264depay ! avdec_h264 ! videoconvert ! autovideosink
sync=false");

            //--------- Enable application extra features
            intent.PutExtra("extraFeaturesEnabled" + camera, false);

            //--------- Add autoaudiosink to featured pipeline
            intent.PutExtra("extraFeaturesSoundEnabled" + camera, false);

            //--------- Scale Video Stream option
            intent.PutExtra("extraResizeVideoEnabled" + camera, false);
            intent.PutExtra("width" + camera, 320);     //used only when
extraResizeVideoEnabled=true
            intent.PutExtra("height" + camera, 200);        //used only when
extraResizeVideoEnabled=true

            //--------- Add plugins
            //ArrayList<String> plugins = new ArrayList<String>();
            Bundle b = new Bundle();
            intent.PutExtra("plugins" + camera, b);
            intent.SetPackage("pl.effisoft.rpicamviewer2");

            StartActivityForResult(intent, 0);
            Button Back_Button = FindViewById<Button>(Resource.Id.VSSBack);
            Back_Button.Click += delegate
            {
                Finish();
            };

        }
    }
}
```

AndroidManifest.xml – Manifest for permissions and application name

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="HovercamMobile.HovercamMobile" android:versionCode="1"
android:versionName="1.0">
	<uses-sdk android:minSdkVersion="16" />
	<application android:label="HovercamMobile"></application>
	<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
	<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
	<uses-permission android:name="android.permission.INTERNET" />
</manifest>
```