Gesture-Controlled Quadcopter System

Anthony Yang Xu, Kendra Crawford, Wenhao Yang

ECE:499 Capstone project

Advisor: Professor Cherrice Traver

March 21st, 2019

Report Summary

As UAV technology advances, it is used in an increasing number of applications. For instance, a quadcopter with a camera could be used by the police to survey the inside of a building to avoid a dangerous situation. However, the complexity of the joystick-button controllers requires a long training time. In addition, the joystick-button controller occupies both hands of the operator. In this project, we will address the problem by designing and building a one-handed gesture-controlled quadcopter. Our design will give police the ability to use one hand for other tasks and an intuitive controller that is easy to learn. In our design, we split it up into three subsystems to be able to solve the specifications for the problem.

The goals and performance of the gesture-controlled quadcopter system are targeted to be able to solve the problem defined above. The first goal is that the system will be lightweight so that it is easy to transport. The system also needs to have less than 5 minutes set up time and will fly for at least 10 minutes. The gestures need to be intuitive this will be measured by the ability to learn how to fly the system in less than an hour. The range from the pilot to the quadcopter must be at least 30 meters and the system must provide video feedback to the pilot so the quadcopter can go out of site.

To be able to solve this problem as stated before we split the system into three subsystems the hand controller, the communication base and the quadcopter. The hand controller reads positional data and special commands from the user and transmits the information to the communication base. The communication base takes the information from the hand controller and converts it to a PPM (Pulse-Position Modulation) signal that can be sent to the quadcopter. The quadcopter receives the PPM signal as a PWM (Pulse-Width Modulation) signal then converts it to an s-bus signal that goes to the flight controller. The flight controller on the quadcopter controls the ESC (Electronic Speed Control) that then controls the motors.

The result was a system that was lightweight and converted the positional data of the pilot's hand to a PPM signal. The video signal was produced by the quadcopter and received by the communication base. The receiver on the quadcopter broke before we could test the full system and we did not have enough time to order a new part, so the full system did not work.

Table of Contents

Initial of the student who wrote the section is after the header

Table of Figures and Tables

1. Introduction

The police force is one of the humble building blocks of any functional society in every country. The main duty of a police officer is to maintain public order and enforce the law when needed. This spans from giving tickets to speeding drivers to handling a situation with violence. Needless to say, the life of a police officer is often dangerous. According to the statistic given by the National Law Enforcement Officers Memorial Fund, 514 police officers casualties have been attributed to gunfire in the last decade(2008-2017)[6]. It is the leading cause of death among police officers and accounts for more than a third of the total 1511 police casualties in the past decade.

In this project, we want to provide a safer solution to police officers that are surveying a building for one or multiple potential dangerous personnel. We are working to design and build a gesture-controlled quadcopter that can scout ahead of the officer and provide information about the area so the officers can avoid a dangerous situation. This gestured-controlled quadcopter system will be compact and lightweight, offering greater mobility and ease-of-use over joystick-based systems. We hope that our system's simple, intuitive design will allow a user to learn how to pilot it in less than an hour. These goals are targeted to create a system that is useful for any police officer, regardless of their technical skills.

This paper will outline the application of UAV (Unmanned Aerial Vehicle) in the police force, and previous work in the field of gesture-controlled quadcopter systems, followed by the ethical and security concerns that come with the use of quadcopters for police surveillance. We will then go over the design requirements that our project must meet to be functional for our application, followed by a discussion of alternative designs we considered. After design alternatives have been explored, the current design of the system and its components will be explained. Since we are building this system, the implementation schedule of the design is then presented.

2. Background

The website Unmanned Aerial wrote an article that said :

> "A growing number of studies show unmanned aerial vehicles (UAVs) are disrupting the status quo across industries. From agriculture and construction to equipment inspection and mining, industry and business leaders are using drones to save time, money and effort, as well as lives." [7]

As stated above, UAV technology is being used increasingly for many different types of fields including the police force.

2.1 Application of UAV in the Police Force

UAV technology is increasingly being used in the police force for many different types of situations, including search and rescue, traffic collision reconstruction, investigating active shooters or suspects, crime scene analysis, and surveillance [10]. As a team, we had informational meetings with people who currently work in the field of UAV, including Ed Capovani from In Sky Aerial, who works closely with the Police Department of Albany. His company actually helps the police perform search and rescue missions in our region[5]. He discussed the difficulty that the police departments, at least in our area, face when it comes to using UAV technology. Even though many police departments have access to the technology, they do not have the time to become familiar with the complicated controls. Also, since they are on rotating shifts, it would be the most effective if all police officers learned how to use the UAV technology, which would take more time in training than the officers have to offer.

2.2 Previous Work in Gesture Controlled Quadcopter Systems

At the time that we began this project, some companies and individuals had already developed gesture controlled quadcopter products. MIT students Ben Schreck and Lee Gross proposed the creation of a gesture-controlled quadcopter using the video supplied from an Xbox Kinect system[13], which would sit stationary and read the movements of the person in the feed to control the quadcopter. There was also a group of students based out of Lanzhou University of

Technology, in Lanzhou, China who submitted a paper outlining the use of a Kinect video sensor in much the same way[16]. The problem is that these designs are not portable. Moreover, Yucan Liu, a computer engineering student who graduated from Union College in 2017, designed a single-handed gesture controller for quadcopters[9]. His goals were for the quadcopter in his system to be able to pitch, yaw, roll and change in altitude smoothly when using his controller. As results for his project, the quadcopter used in his system was able to respond to yaw, pitch, and roll commands within 0.3 seconds. The project we are working on is based on Yucan Liu's project.

Among all the gesture controlled quadcopter products we researched into, the mostly used control instructions are listed as follows: tilting the hand forward, backward, to the left and to the right to instruct the quadcopter to fly forward, backward, to the left and to the right. Lifting the hand vertically up or down to increase or decrease the altitude of the quadcopter. For example, a company called KD Interactive USA has produced a toy called Aura [2] that takes the hand movements above to guide the quadcopter. Thus, we are building up our own control gestures using the popular gestures for quadcopter controlling as a basis. Aura also supports auto take-off, auto-hover, 360-degree flip, head lock mode, and auto landing, which inspired us when designing our own special commands for combinational movements.

2.3 Potential Project Impacts

Since our project is intended for police departments, the questions of privacy and security are two important aspects we have to consider for a final product. When researching drone laws, we found that one is prohibited from taking photos or videos of others without their consent anywhere they have the right to expect privacy[15]. Since this system would be used by the police department it would have the same rights as the police officers when it comes to invading privacy, which is outlined by the 4th Amendment :

> "The right of the people to be secure in their persons, houses, papers, and effects, against unreasonable searches and seizures, shall not be violated, and no warrants shall issue, but upon probable cause, supported by oath or affirmation, and particularly describing the place to be searched, and the persons or things to be seized.[8]"

This means that the police would not be able to use the quadcopter on a person’s property without first getting a warrant or having probable cause of illegal or dangerous activities happening within the property.

Since the quadcopter will be used specifically for indoor surveillance there are not many quadcopter laws that apply to it. The applicable laws detail having to register the quadcopter and needing to take a quadcopter safety training course before being able to use it in the field. Other laws that apply to quadcopter all have to do with visibility and making sure that the quadcopter does not interfere with other aircraft vehicles. When it comes to our actual system, security measures will have to be included to protect the system from receiving interfering signals.

3. Design Requirements

In order for this project to function efficiently and serve its purpose well, it needs to satisfy requirements that are defined by the problem. As we had discussed in previous sections, the controller needs to be intuitive to operate because the police officers cannot afford to have long training sessions. The controller and the drone need to operate together and transmit live video in an indoor setting up to a range of at least 30 meters to ensure a versatile usage. The controller and the drone should be lightweight so they do not hinder the movement and carrying capabilities of the officers in action. Last but not least, the setup time for the system must be within five minutes since timing is always important in every emergency situation.

We decided to divide the project into three parts: the gesture controller, communication base, and quadcopter. Our design separates the controller and communication base to minimize the weight of the gesture controller, thereby minimizing the fatigue and reduction in mobility of the wearer. Instead, the communication base will do the “heavy lifting” in terms of calculation and interpretation of the gesture and data transmission between the controller and quadcopter. In the following paragraphs, we will elaborate on the requirements for each part.

3.1 Hand Controller

The hand controller is what most greatly sets our system apart from traditional joystick-based quadcopter systems. This controller offers a very intuitive, simple method of piloting such that those without any experience with UAV can fly it within an hour of receiving the system. The hand controller needs to be able to run for at least ten consecutive minutes without requiring a battery change, so the system must function on very little power to avoid a bulky battery. The hand controller will be able to fit most adult human hands and will be robust enough to fall from 1.5 meters without breaking. Any setup for the hand controller, which will include calibrating the motion/position sensor, will take less than 5 minutes, as speed is of great significance in an emergency situation. The controller will weigh less than 250 grams so as not to fatigue the pilot while they are controlling the quadcopter. There will be specialized commands such as take-off, stop, and hover, which will simplify difficult quadcopter functions. While the pilot is controlling the quadcopter the specialized commands, which are indicated by buttons and a flex sensor, should take priority over the movement controls. For example, when the quadcopter is in hover mode the pilot will be able to move their hand without sending the movement to the quadcopter. Most importantly, the hand controller must be able to take information from the movement/position sensor, buttons, and flex sensor and convert it into strings, which fit the decided format, then send it over radio communication at 433 MHz to the communication base where it will parse the information sent.

3.2 Communication Base

The main goal for the communication base is twofold. First, as the name suggests, the base is responsible for communication between the other parts of this system. Secondly, the communication base needs to calculate and interpret the motion data and translate it into the corresponding control signal for the quadcopter.

The communication base needs to handle three lines of communication, that is from the hand controller to the communication base, flight control from the communication base to quadcopter and video data from quadcopter back to communication base. Since an operator will wear both the controller and communication base, the communication range needed is 5 meters at most. However, the communication between base and quadcopter is more complicated. The quadcopter needs to operate at a range of at least 30 meters indoors from the communication base in order to scout and survey efficiently. For the operator to get visual feedback, the live-stream video transmission needs to have a similar range. In addition, the resolution for the camera needs to be high enough in quality for the operator to distinguish the age, gender, and potential weapon carried by the suspect, especially in dim light. This requires the camera to have at least a resolution of 960p for a digital camera or 900TVL(TV Line) for an analog camera. TV Lines are a measure of horizontal resolution[14]. The delays from the controller to the quadcopter and from camera to base need to be shorter than 0.6 seconds, the minimum human reaction time.

The communication base needs to interpret and decode the motion data from the gesture controller. The decoding time would contribute to the overall delay between gesture and flight command, thus the hardware needs to have a fast processing speed and the algorithm needs to be efficient. For now, we are estimating the processing time should be less than 0.3 seconds. This timing would not be hard to keep for basic hand movement, but definitely will be more challenging as we add in more complicated gestures.

Finally, since the communication base will be carried by a police officer in action, it can not be cumbersome or heavy. Rather, it should be a small box that can be hooked on a belt. In order to achieve this, the communication base should be smaller than a 10x10x5cm and lighter than 4 kilograms. The maximum operating time for the base should be at least an hour.

3.3 Quadcopter

The quadcopter will be piloted using the hand controller and it will live-stream video back to an FPV (first person view) monitor. In order to accomplish those tasks, the quadcopter should meet the following requirements.

First, the gesture controlled quadcopter should be able to fly just like a traditional quadcopter and support basic quadcopter movements: roll, pitch, yaw, and throttle. In addition, the quadcopter should also be able to conduct special movements such as emergency stop and take off. With the special commands, the gesture controlled quadcopter system can be more intuitive and safer to fly.

Second, the quadcopter should respond to any instruction from the gesture controller within 0.6 seconds when it is within the operating range from the communication base. The short response time allows the pilot to have more precise control over the quadcopter.

Third, since high-speed UAV crashes can cause severe injuries, the maximum speed of the quadcopter in this system will be limited to 10 meters per second. This will allow the quadcopter to fly fast enough for its intended application while not being dangerous to people.

The body of the quadcopter should be smaller than a $25x25x25\,cm^3$ so that it will be portable enough for a police officer to carry in his or her bag. The weight of the quadcopter will be less than 1.5 kilograms to maximize its agility and minimize power consumption. To survey the inside of a building, 10 minutes would be sufficient; thus, each battery pack will is estimated to require at most 3 hours to be fully charged and support at least 10 minutes of flying time.

Finally, it will take no more than 5 minutes to pair the radio receiver on the quadcopter with the radio transmitter on the communication base, providing easy setup between the quadcopter and the rest of the system.

Specification	Value	Unit
Hand Controller		
Basic Movements	Gestures Measured by Sensors	-
Special Movements	Emergency Stop; Takeoff, Hover	-
Weight	250	Grams
Falling Height	1.5	Meters
Communication Frequency	433	MHz
Fly time	10	Minutes
Set up Time	5	Minutes
Communication Base		
Operating range with	5	Meter

controller		
Operating range with quadcopter	30	Meter
Video Resolution	900	TVL
Latency	0.6	second
Weight	4	Kg
Size	10*10*5	cm
Quadcopter		
Basic Movements	Yaw; Row; Pitch; Throttle	-
Special Movements	Emergency Stop; Takeoff	-
Instruction Responding Time	0.6	second
Maximum Speed	10	meter per second
Volume	25^3	cubic centimeter
Weight	1.5	kilogram
Operating Time	10	minute
Charging Time	3	hour
System Setup Time	5	minute

Table 1. Design Requirements for the System

4. Design Alternatives

In this project, there are multiple choices for each component that would allow the quadcopter system to achieve the desired functionalities. For example, both of the radio based and the bluetooth based communication modules would allow the communication between the subsystems. In this section, each component choice for the most critical components will be evaluated by how well the system built with it will achieve the design requirements.

4.1 Hand Controller

There are several ways people have implemented similar gesture-controlled systems. Some have done it with video recognition such as the students who worked with the Kinect sensor [16], some have done it using proximity sensors and others have done it with a glove-like controller as we plan to do. We started with a glove design because this project started out as a continuation of a past student Yucan Liu's senior project. This makes sense in the use-case of the police force in the field. Once the hand controller's general design was selected, we broke the design down into four major component decisions: the motion/position sensor, the microprocessor to read and convert the signals, and the specialized command indicators.

The most important component is the motion/position sensor because this is what the quadcopter's movement is relying on. The criteria for the position sensor include the ability to sense the hand's position and speed, being low powered, not being larger than 5x5 cm^2 and compatibility with Arduino. To define compatibility with the Arduino program is to have a library that can be downloaded and used in the program. This will allow the communication between the program and the sensor to be simple then the focus can be on interpreting the data. An example of a function that would be useful is a data reading functions since most motion sensors are recording multiple types of data it can be sent over a bus and having a function that can read the information from the bus and parse it into readable data such as a string is very useful. The table below shows the breakdown of three alternative components and the sensor that was chosen:

Component	Senses Position and Movement (40)	Low Powered (20)	Smaller than 5x5 cm^2 (20)	Works with Arduino (20)	**Total (100)**
KIONIX - Accelerometer	20	20	20	20	**80**
NXP Semiconductors - MMA8453QT	20	20	20	20	**80**

Accelerometers Sensors					
Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout	**40**	**10**	**18**	**20**	**88**

Table 2. Decision Matrix for Motion/Position Sensor for the Hand Controller

The Adafruit 9-Degrees Of Freedom Breakout sensor was the one that was chosen because it is able to sense the position relative to the strongest magnetic field, which is normally true north, acceleration in 3D space, and rotation measured from the center of the board of the object it's attached [1]. Having both the position and motion allows for more precise control of the quadcopter because both can be used in computing the direction the quadcopter should go in.

The next major decision was the "brain" of the hand controller. The important criteria for the microcontroller to meet are the ability to communicate with the Adafruit 9-DOF sensor, having a way to be powered without a bulky battery, having a way to have secure wiring without using a breadboard, and having a compatible device library for the Arduino IDE. The table below shows the breakdown of three alternative components and the microcontroller that was chosen:

Component	Easily Communicates with the sensor (40)	Powered without Bulky Battery (10)	Wiring other than a Protoboard (30)	Library in Arduino (20)	**Total (100)**
Adafruit Trinket - Mini Microcontroller	0	10	15	20	**45**
Sparkfun AVR Pro Micro	30	5	15	20	**70**

LilyPad USB Plus	**40**	**10**	**30**	**20**	**100**

Table 3. Decision Matrix for Microcontroller for the Hand Controller

The LilyPad USB Plus is the best choice for a microcontroller for the hand controller it has an Arduino library and the required SDA and SCL pins to work with the Adafruit 9-DOF sensor. The other interesting aspect of the LilyPad is that it works with conductive thread so the entire circuit can be sewn into the fabric of a glove reducing the bulkiness of having a soldered breadboard. The LilyPad USB Plus is also a low powered microcontroller that can be powered by a separate LilyPad compatible coin cell battery holder.

One of the final decisions of the hand controller was what would control the specialized command signals. The components had to be low powered, simple, and attachable to the conductive thread. These criteria eliminated components like a touchpad on the palm of your hand because it would be difficult to attach to the conductive thread and the pressure required to trigger it could fatigue the pilot's hand. The final choice was a flex sensor along a finger to trigger the Take Off function and LilyPad buttons to trigger the Hover function and the Stop function. The final aspect of the hand controller that was considered was the communication between it and the communication base that will be discussed more in the communication base section.

4.2 Communication Base

The most important decision for the communication base is the selection for transmitter and receiver. The main consideration for communication would be latency, type of data input/output, range, power consumption, and price. For the communication between the gesture controller and the communication base, the operating range is a lesser concern as the two parts will work in a close range. On the other hand, the latency and power consumption are more important because we want to minimize the overall delay and the glove is not capable of supporting a lot of power. The two potential choices are using a short-range radio communication module or Bluetooth module. The strength and weakness can be found in the following table:

Component	Latency (20)	Data Reliability(25)	Range (10)	Power Consumption (25)	Price (20)	Total (100)
433Mhz RF Transmitter and Receiver Module[433]	15	25	8	22	18	88
Bluefruit LE - Bluetooth Low Energy (BLE 4.0) - nRF8001 Breakout - v1.0[bluetooth]	15	20	8	25	12	65

Table 4. Decision Matrix for the Communication between the Hand Controller and Base

Low power Bluetooth is great in terms of power consumption, but it suffers from having less reliability in terms of data transmitting and the price is more than double the price of the 433Mhz radio module.

The requirements for the communication base to quadcopter has different requirements. Both the communication base and quadcopter will have a larger power supply that means the power consumption will be less of a concern. The latency and range indoor becomes our major concern as the quadcopter needs to navigate across rooms and move outside the line of sight.

In the world of quadcopters, 2.4Ghz radio communication is used most frequently as it offers a good range of operation and penetration without requiring a huge antenna to receive the signal. Once we narrow our scope to 2.4Ghz radio communication, we have two transmitters to choose from, hobby level DIY transmitters and Xbee. The hobby level DIY transmitter is a specialized transmitter that is tailored for flying hobbyist quadcopters. We can use the configuration and performance that are preset on the transmitter without too much work. The downside is that we need to decode the mapping between the input signal and the output signal. Xbee is an industry level wireless communication that can be fully configured by the users. This opens up a lot of availability and potential for our project. However, since our focus is on gesture control, we are not going to use Xbee since the work of building a control system for quadcopter from the ground up is too much and overthrows the benefit that Xbee can offer. We pick FrSky DHT 8ch since it is the only available DIY transmitter module on market.

After the transmitter is being picked, we can narrow our selection of receiver as there are only a handful of receivers that are compatible with it. The following table shows the options:

Component	Latency (30)	Data Feedback (20)	Range (30)	Power Consumption (10)	Price(10)	Total (100)
FrSky D8R-II plus	25	20	25	8	8	86
FrSky V8FR-HV	25	0	25	8	8	66

Table 5. Decision Matrix for the RC receivers between the Quadcopter and Base

The D8R-II is better than its counterpart because it can send the status of the quadcopter back to the transmitter on time, a feature called telemetry. Telemetry enables us to monitor the status of the quadcopter during flight, which can valuable in multiple scenarios. V8FR-HV does not have telemetry, which means we can not receive feedback at all during the flight.

The third type of the communication is the video stream from the camera mounted on the quadcopter to the monitor included in the base. In this project, we are using the already established hobby level FPV(first person view) camera. This special kind of camera can be connected to video transmitters and broadcast the image via radio signal. The signal will be received by an FPV monitor as long as the video transmitter and the monitor are on the same channel. For the monitor we picked the cheapest 7’’ monitor on the market since the main difference among the monitors is the resolution. The selection of camera and video transmitter will be discussed in the last part of the quadcopter section.

We picked an Arduino Uno as the base microprocessor that would analyze the input command and output the correct radio transmitter signal. Although there are faster and smaller microprocessors than Uno, it is a good starting point as it offers enough processing power with its 16Mhz clock speed. Uno also has 14 digital pins and 6 analog pins that enable further potential expansion for the communication base. More importantly, it is the most used microprocessor in Arduino microprocessor family so a lot of libraries are available for the use of this project.

4.3 Quadcopter

In order to design a quadcopter that meets the design requirements, the quadcopter should at least have the following components: frame, flight controller, four ESCs (Electronic Speed Controllers), four rotational motors, four propellers, a lipo battery, a lipo battery charger, a FPV camera, a FPV video transmitter, a radio receiver and a signal converter. For each component, there are a fair number of alternative choices in the market. Thus, the design choices for the pivotal components are described below.

The flight controller is the most important component which works as the “brain” of the quadcopter. More specifically, it translates the radio signal from the radio receiver to PWM (Pulse Width Module) signals and pass them to the ESC and motors to actuate the desired movements of the quadcopter. To evaluate how well a flight controller fits in this system, there are six major criteria: (1)if allows source code modification, (2)if it supports an FPV camera, (3) safety, (4)responsiveness, (5)availability of resources, and (6)cost. Among the criteria, if supports source code modification and if the quadcopter supports an FPV camera are the most important criteria. Normally, a flight controller is pre-programmed to support basic movements of a quadcopter. However, in order to conduct special movements command and limit the maximum speed of a quadcopter, modifications have to be made to the source code installed in the flight controller. Thus, for the desired application, it is important for us to have access to modify the source code of the quadcopter to implement special movements such as take-off. The responsiveness and available resources are the second most important criteria.

The responsiveness is important since the quadcopter has to respond to a instruction within 0.6 seconds. The availability of resources is also important since resources such as exhaustive user manual and guidance in different formats allow us to have a comprehensive understanding of the target component. If there is very limited resource for a flight controller, there is a much larger chance of failure of the system. Based on the importance of each feature of the desired flight controller, the weights for each criterion are shown in the table below.

Product	Source Code Modification (25)	FPV camera (25)	Responsiveness (15)	Availability of resources(15)	Safety (10)	Cost (10)	**Total Score (100)**
DJI Naza MV 2	0	25	12	8	10	2	**57**
3DR Pixhawk Mini	25	25	12	8	10	3	**83**
Kakute F4 V2 AIO	25	25	12	14	6	8	**90**

Table 6. Decision Matrix for Flight Controller for the Quadcopter

For the flight controller, three popular models in the market are considered: DJI Naza MV 2 [11], 3DR Pixhawk Mini [12], and Kakute F4 V2 [17]. In terms of modifiable source code, the 3DR Pixhawk Mini and Kakute F4 V2 AiO allow customized programs, but DJI Naza MV 2 does not allow the user to customize installed code. Thus both Kakute F4 V2 AIO and 3DR Pixhawk Mini get 25 points for this criteria, and DJI Naza MV 2 gets 0 points. All of the flight controller choices have ports for FPV camera, so all of them get 25 points for the FPV camera criteria. In terms of responsiveness, all of the flight controller choices are widely used for racing quadcopters, and in this way, they all get 12 points out of 15 points. For the next criteria, availability of resources, all alternatives have a well-documented user manual, but only Katute F4 V2 AIO has a great number of tutorials written by hobbyists. Thus the Katute F4 V2 AIO gets 14 points and the others get 8 points. For safety aspect, both DJI Naza MV 2 and 3DR Pixhawk Mini have predefined functions to avoid the quadcopter from crashing, thus they both get 10 points. However, for Kakute F4 V2 AIO, it allows the user to customize the source code, users can have their own functions to ensure safety thus, it gets 6 points. In terms of cost, DJI Naza MV 2 costs $159, 3DR Pixhawk Mini costs $138, and Kakute F4 V2 AIO costs $41. Thus, the alternatives described above get 2, 3 and 8 points for the cost criteria respectively.

The second design choice to make is the ESC (Electronic Speed Controller). There are a lot of available quadcopter ESCs in the market such as the 30A ESC Simonk, DJI E310, and Racerstar 30A V2. All of them are powerful enough to support race quadcopters. In terms of compatibility, DJI E310 is mostly used to work with DJI flight controller, while both Simonk and Racerstar are widely used for a broader range of flight controllers. Thus, Simonk and Racerstar could be the better choices. In terms of price, Racerstar is $14 each and Simonk is $25 each. This price difference makes Racerstar the preferred component.

The third design choice to make is the FPV camera and video transmitter. The main requirement for the FPV camera is resolution and performance in dim light since many potential scenarios for this project is for indoor use without light. In the world of analog camera, the unit to measure resolution is TV line or TVL. 900 TVL roughly equals to 960p for a digital camera. We need a camera with at least 900 TVL in order to distinguish the face of our target and the potential weapons carried at a range of 5 meters. In order to have a tolerance for low light, we picked a 1200 TVL camera from Foxeer, which is Foxeer Falkor.

For the video transmitter, we picked EACHINE VTX03 due to its small size, and balance between its power consumption and range. One major benefit of VTX03 is the ability to switch to four different power consumption modes. A higher power consumption means a better transmission range, but that would drain the battery faster and would potentially overheat and damage the circuit. The ability to switch power consumption gives us a better solution under different situations.

For the frame of the quadcopter, the frame Frame Wheel F450 and frame Martin II are considered. Both frames can mount an FPV camera and are robust and lightweight. However, the F450 costs $33 while Martin II costs $27. Thus, the frame Marin II is preferred.
For the rest of the components, they have less interesting design alternatives to discuss, and most of the alternative component choices are rated in terms of compatibility with other components and cost. The full list of components chosen can be seen in Appendix 1.

5. Preliminary Proposed Design

The gesture-controlled quadcopter system will have three parts: the hand controller, communication base, and quadcopter. The hand controller will detect the gestures of the pilot and send them to the communication base. The communication base will then convert the data that it receives from the hand controller into sendable instructions for the quadcopter, then it sends the data to the quadcopter. The quadcopter responds to the information sent from the communication base and moves in the correct manner. The quadcopter also takes live video data from an FPV camera and transmits it back to the communication base's monitor.

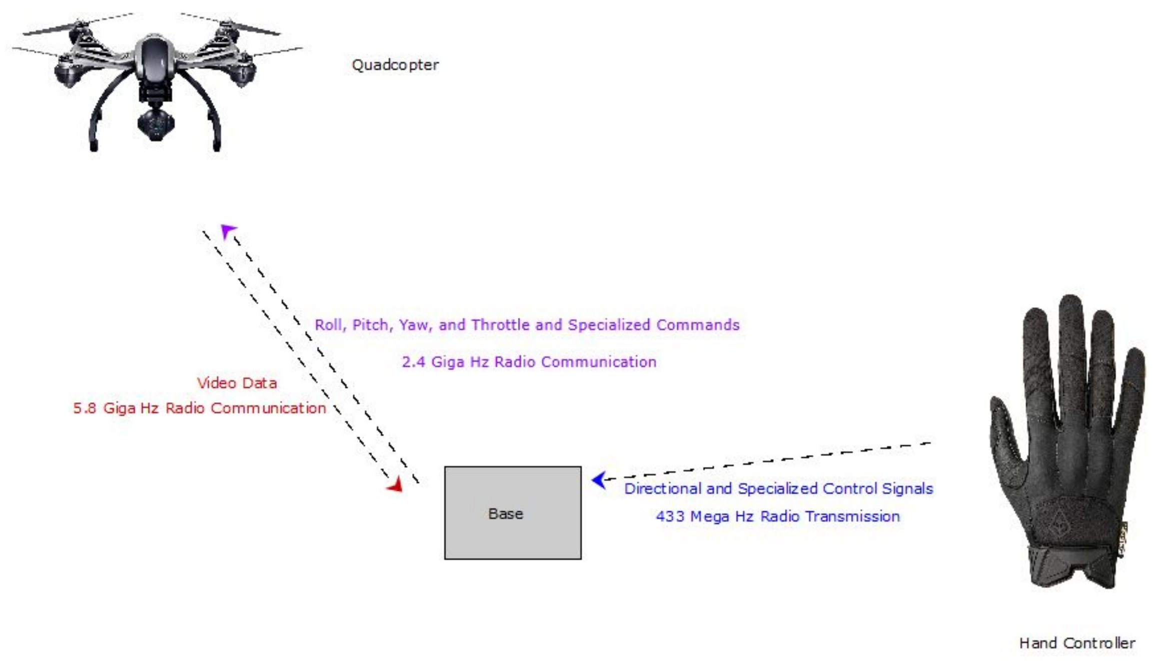

Quadcopter
Roll, Pitch, Yaw, and Throttle and Specialized Commands
2.4 Giga Hz Radio Communication
Video Data
5.8 Giga Hz Radio Communication
Base
Directional and Specialized Control Signals
433 Mega Hz Radio Transmission
Hand Controller

Figure 1.The Top-Level Diagram for the Gesture-Controlled Quadcopter System

5.1 Hand Controller

5.1.1 General Design

The hand controller is broken up into three general tasks, one is reading the hand's input this includes the position and motion, and specialized commands. The second task is converting that into a string that the communication base can read. The final task is to take the string with the data, whether that be a specialized command or a motion reading, and send it to the communication base via radio communication. Below is a block diagram showing the system and the types of communication between them.

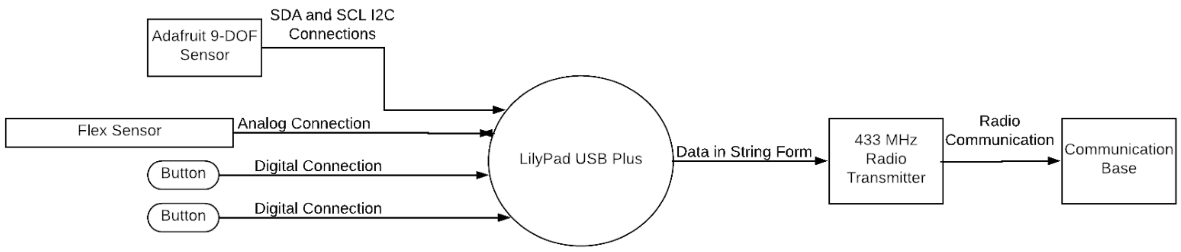


Figure 2. The Block Diagram of the Hand Controller

The actual hand controller will be sewn, using conductive thread, into a glove that stretches so most hand sizes can control the quadcopter. The buttons will be placed on the palm so the thumb can reach them to press them. The flex sensor will be lining the middle finger with the hope the glove will be able to be used on either hand. The LilyPad, the 9-DOF sensor, and the radio transmitter will be on the back of the hand to not impede motion. How the hardware is actually connected is outlined in the next section.

5.1.2 Gesture Design

As described in the background of the paper there has been companies that have created other gesture controlled quadcopter products. The one that was thoroughly discussed the the Aura produced by KD Interactive USA [2]. This product uses a hand controller band that slips on to the hand and then the pilot can do the movements outlined in the manual to move the quadcopter. We have decided to base our gesture design on this product's controls. They are outlined in the table below.

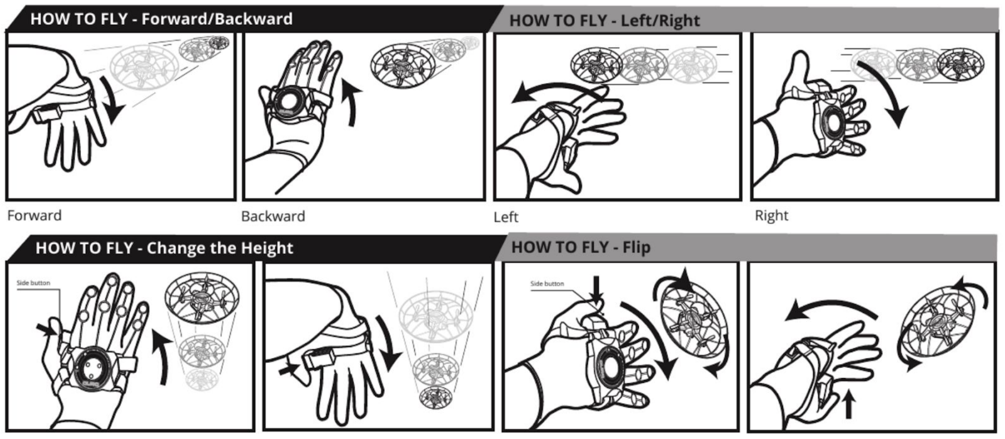


Figure 3. Hand Gestures Instruction for Aura

Though there will be some key differences between our gesture design and the picture from the manual above. The first being our quadcopter will not have a flip function, for our purposes the flip function is not needed and could cause disorientation to the pilot when using the camera. Secondly, as you may have noticed the up and down controls and the same as the front and back controls and a button is used to differentiate them. In high stress environment like the one we are trying to design this system for the lack of differentiation could cause the pilot to mistake one direction for another and crash the quadcopter. Instead of having the two commands be so similar the pilot will move their hand up and down to move the quadcopter. We will adjust the movement of the pilot to be amplified so the quadcopter has more range than the pilot. Also we have the enable button which allows the pilot to keep the quadcopter hovering in the place it was left and the pilot will be able to readjust their hand position.

5.1.3 Hardware Design

As mentioned in the previous section the wiring for the system will be done with conductive thread so the components can be sewn into a glove for ease of use and transportation. The brain of the system is the LilyPad USB Plus this is the microcontroller that is going to read the sensors and also facilitate the communication between the communication base and the hand controller. The actual microprocessor running in the LilyPad USB Plus is an Arduino Mega this means it's compatible with Arduino and Adafruit products [4]. The Arduino microprocessors have SDA and SCL pins that allow for easy communication between the processor and the sensor, all that is needed is a downloadable library from the Adafruit website [1]. The SDA and SCL are the data and clock line respectively for I2C communication. The library provided by Adafruit for the SCL pin setup means that the programmer will not have to set up the clock for communication manually. The other two types of inputs are the flex sensor and the buttons. The flex sensor also has a downloadable library for the Arduino IDE and the connection type is a simple analog reading pin. The buttons also are designed to work with LilyPad products and only require a simple digital input pin when setting up the circuit. The final part of the circuit is the radio communication transmitter this requires a digital output pin. The schematic for the hardware components of the hand controller can be seen below:

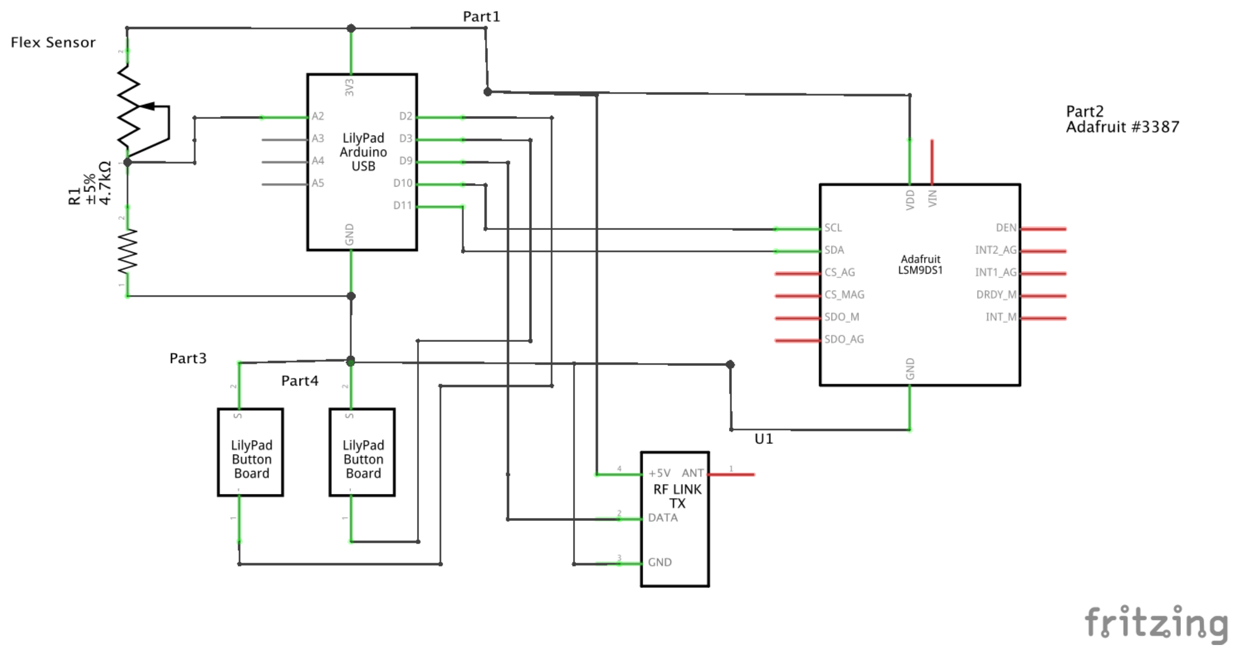


Figure 4. Schematic of the Hand Controller

5.1.4 Software Design

The program being used design the software as discussed in the design specifications section is Arduino. Arduino communicates with the LilyPad USB Plus by using a downloadable library and also had built in libraries for all of the components which need one. Below is the outline of the software design, this includes the continuous functions and the specialized commands that the hand controller will be performing. The continuous functions include reading the data from the 9-DOF sensor which will be done using a library function called lsm.getEvent(), and then the controller will take this information and send it with the 433 MHz radio transmitter. The transmitter takes a buffer that has strings in it, the communication can be unreliable so the messages will be short. The continuous functions will continue happening or won't start until a specialized command is sent.

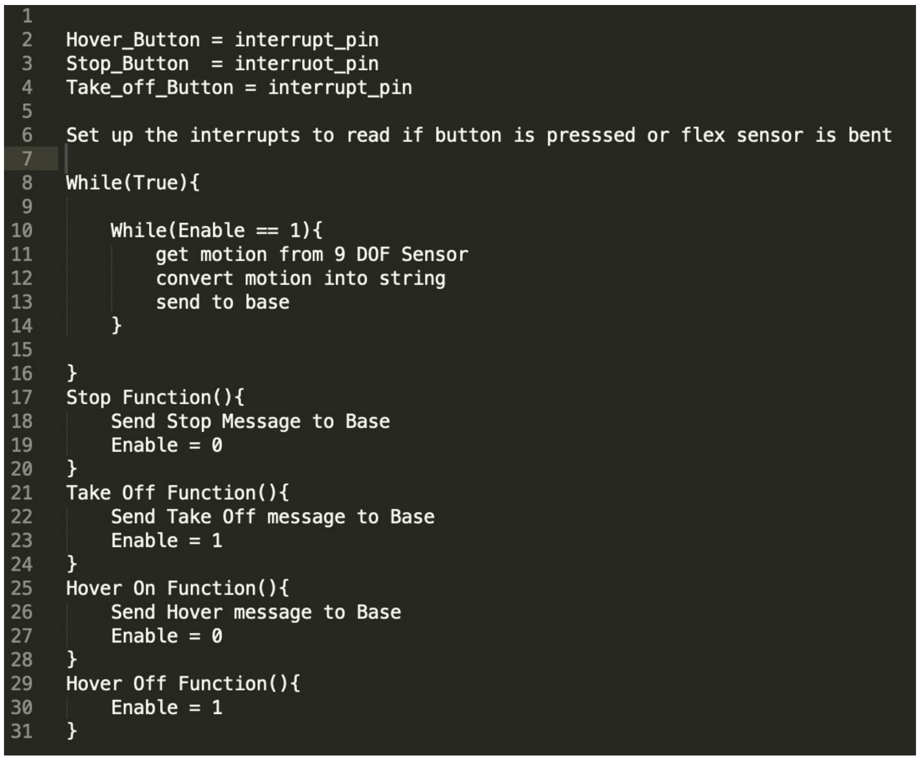
The specialized commands will be triggered by using interrupt pins for the two buttons and making an interrupt in the software for the analog read for the flex sensor. The specialized commands have the priority of Take-Off, Stop, and then Hover. Take-Off has the highest priority because it is what starts the quadcopter initially and after the Stop command has been given. Though if the Take-Off command is pressed during flight it would not affect the quadcopter. The Stop command is the second highest priority this is because we want the quadcopter to stop what is it doing during the flight and land for safety reasons. The Hover command is the interrupt with the least priority because it is non-essential to the safety of flying the quadcopter. The reason it is also a programmed interrupt and not a hardware interrupt is because it requires an analog pin, which cannot trigger an hardware interrupt. The software based interrupt will react slower than a hardware based interrupt because it will depend on the system reading the pin and then processing the change that the value makes on it. All of the specialized command are going to control one variable which is called enable. When enable is equal to one the hand controller will send the position information to the radio communication when it is equal to zero the hand controller will not send the information. The Take-Off function will set the enable pin to one and send a special command to the base to tell the quadcopter to take off then wait for hand movement controls. The Stop command will turn enable to zero and send the specialized command to land the quadcopter. Finally, the Hover when initially pressed will set enable to zero and send the specialized hover command to the communication base, then when pressed again it will set enable to one and start sending the position data from the 9-DOF sensor again. The software design outline and pseudocode are described below.

Hand Controller

- Periodically sends the motion to the base
    - Receives input from the 9-DOF sensor, Pin Type: I2C protocol (SDA and SCL)
    - Uses Arduino library to get the information from the sensor
    - Converts the information into a sendable String
    - This is cyclic and does it every couple of ms
    - Controlled by an enable variable
- Hover Button

    - Interrupt message, Pin Type: Digital I/O
    - A specialized message that is sent to the base that tells the quadcopter to hover if it is currently in the air
    - Signal LED on the microcontroller to show the controller is disabled
    - Movement of the hand will not be sent to the controller until it is taken out of Hover mode
    - The quadcopter can be instructed to land from this mode
- Stop Button
    - Interrupt message, Pin Type: Digital I/O
    - A specialized message that is sent to the base that tells the quadcopter to land if it is currently in the air
    - Will disable the movements of the controller
- Take Off Flex Sensor
    - Interrupt message, Pin Type: Analog
    - Since the data is Analog there will be a threshold of “bentness” that will indicate the start function
    - This function enables the hand controller to read the motion of the hand after the quadcopter has finished the task
    - This will start the quadcopter with an initial takeoff function that will go to up about meter then hover

The pseudo code that implements the functions described above is listed below:



```

Hover_Button = interrupt_pin
Stop_Button  = interruot_pin
Take_off_Button = interrupt_pin

Set up the interrupts to read if button is presssed or flex sensor is bent

While(True){

    While(Enable == 1){
        get motion from 9 DOF Sensor
        convert motion into string
        send to base
    }

}
Stop Function(){
    Send Stop Message to Base
    Enable = 0
}
Take Off Function(){
    Send Take Off message to Base
    Enable = 1
}
Hover On Function(){
    Send Hover message to Base
    Enable = 0
}
Hover Off Function(){
    Enable = 1
}
```

Figure 5. The Pseudocode Describing the Functionality of the Hand Controller

5.2 Communication Base

5.2.1 General Design

There are four sub-functions in the communication base section. The first one is accepting controlling information from the hand controller via the 433MHz radio communication receiver. The Arduino Uno in the base will interpret the command and convert it into corresponding flight command. The third one is to send the flight command to quadcopter via the transmitter. The last part is to use the monitor to accept and display the video signal from the quadcopter. The high-level hardware connection is shown below：

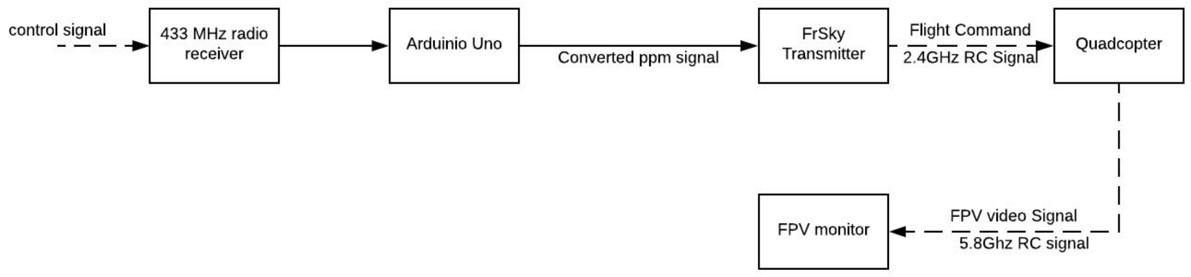


Figure 6. Overall Hardware Diagram of Communication Base

The final product for the communication base should be soldered and contained in a closed box with a size roughly of 50x50x50 cm^3. There will be an intuitive user interface to turn on power and bind the transmitter with the receiver on the quadcopter. The monitor will be a separate component as it does not require any physical connection with other components in the communication base.

5.2.2.Hardware Design

The physical wiring is simple as the data flow is one way and there is no feedback. The hardware is composed by only four parts: a 433 MHz radio receiver, an Arduino Uno, a FrSky DHT 8ch DIY radio transmitter and a FPV monitor that is isolated from the previous three parts. The data flow is as follows: the controller command is sent to Uno via 433 MHz radio signal received from the radio receiver. The command is interpreted by the Uno and translated to the corresponding flight command. The flight command is transferred to the FrSky transmitter in form of PPM signal and transmitted to the quadcopter in 2.4GHz radio signal. The FPV monitor receives the 5.8GHz radio signal from the camera mounted on quadcopter and displays the video. Below is a pin diagram for the hardware layout. Note the symbol for the receiver is not the correct one for FrSky DHT 8ch DIY as the correct symbol is not included in the library of the drawing software.

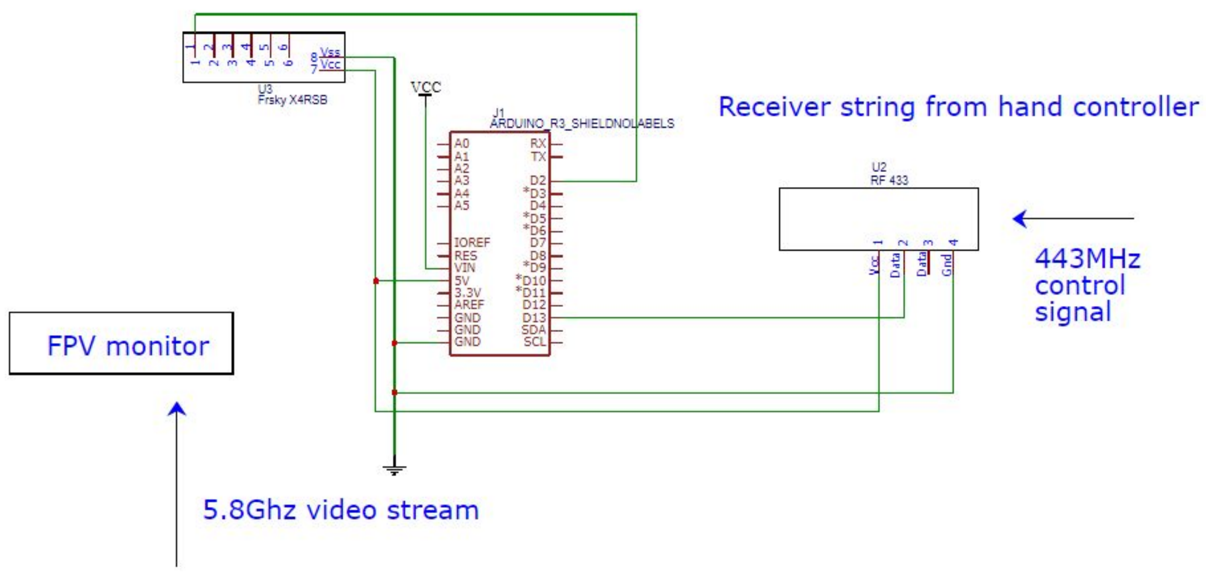


Figure 7. Pin Diagram of Communication Base

5.2.3 Software Design

The software development will be focused on processing the incoming data from the gesture controller and passing the interpreted flight command to the transmitter. The information from the gesture controller consists of two types: motion data collected from a 9-DOF sensor and special commands such as land, hover and takeoff. If the input is motion data, the processes will try to interpret the meaning of the gesture and send corresponding flight command. If the input is a special command, a preset command will be sent to the transmitter.

However, after the Hover and Land commands are sent to the base, the hand controller will stop outputting command until a different command is called. This requires that the communication base be able to memorize the last command given. Thus the overall software structure in the communication base will take the form of a finite state machine. There are three states in the code: landing state, hover state and flight state. The three states will be tracked by two boolean variables: land and hover. If hover == 1, the code is in hover mode. If land == 1, the code is in land mode. If both variables are 0, the code is in flight mode. There is no situation were the two variables can be both equal to 1. The state transfer diagram and the pseudo code are shown below.

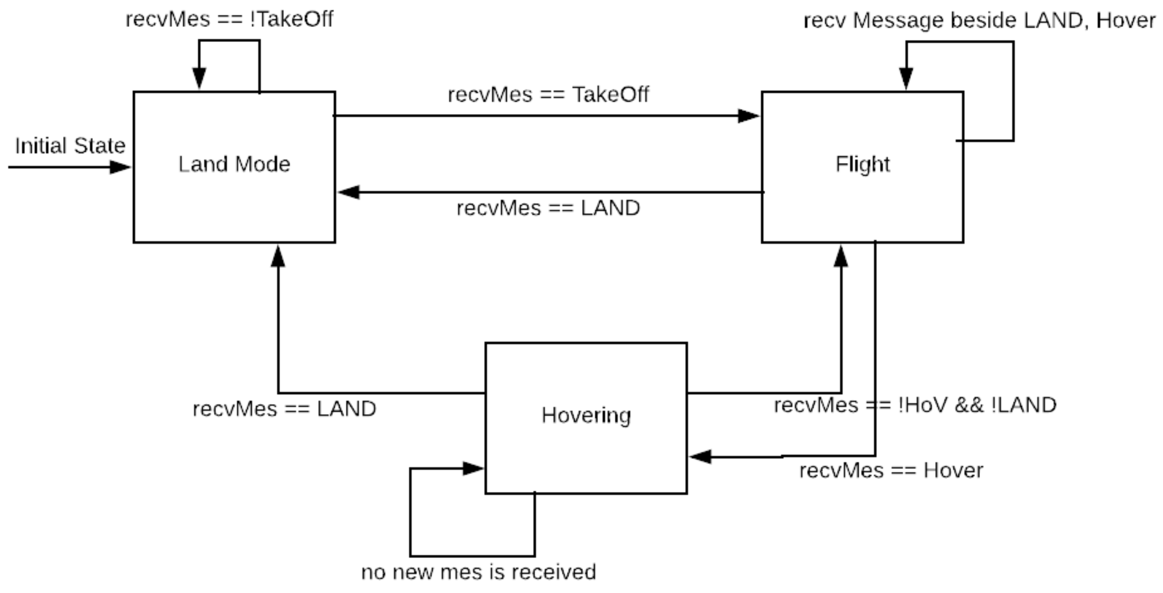


Figure 8. States in Software Level of Communication Base

```
Base_mainsetup()
$initiateReceiver()$
$initiateTransmitter()$
$//initiatethestartingstateinLandstate$
$Land = 1$
$Hover = 0$

Base_mainloop()
while True
    while Land
        Get input String
        if $input == "TakeOff"$
            $takeoff()$
            $Land = 0$
    while HOVER
        Get input String
        if $input == "LAND"$
            $Land()$
            $Land = 1$
        else if $input == GestureData$
            $gestureInterpretation()$
            $HOVER = 0$
    while $!Land$ && $!HOVER$
        Get input String
        if $input == "LAND"$
            $Land()$
            $Land = 1$
        else if $input == "HOVER"$
            $Hover()$
            $Hover = 1$
        else
            $gestureInterpretation()$
```

Figure 9. Pseudocode of Communication Base Main Loop

The pseudocode above shows the high-level implementation of the main loop in the base. Under land and hover mode the code will only change output if certain instructions are given. The basic functions we are going to use in the base are:

- stringIn(): take in the input 433MHz radio signal and convert it into a string
- initiateReceiver(): initiate the 433MHz radio receiver
- initiateTransmitter(): initiate the FrSky DIY 2.4 GHz radio transmitter
- gestureInterpretation(): Interpret the given motion data, convert it into the ppm signal.
- land(): Send the PPM signal that instructs the quadcopter to land
- hover(): Send the PPM signal that instructs the quadcopter to hover

5.3 Quadcopter

5.3.1 General Design

The desired behavior of the quadcopter in the gesture-controlled quadcopter system is to fly according to the signals received from the communication base and live stream video data to the FPV monitor. Thus, there are two paths of data flows within the quadcopter. The first is the control data flow. The control flow starts from the 2.4 GHz radio signal received from the communication base. Then the radio receiver module forwards its PWM data to PWM-SBUS converter, which converts a PWM signal to a Sbus signal and passes the signal to a flight controller. Once the flight controller receivers the Sbus signal from the converter, it will decode the data and send four PWM signals to four ESCs(Electronic Speed Controls) that are connected the flight controller, and each ESC will generate signals to the motor connected to it. In this way, the actuation of the flight instruction from the communication base is accomplished. The second data path is designed to allow the FPV camera on the quadcopter to live stream video data. Thus, the data flow starts from the output of the FPV camera. Once the FPV camera has generated video data, it sends the data to the flight controller. When the flight controller receives the video data, the flight controller forwards it to the FPV transmitter. Once the video content reaches the FPV transmitter, it will be carried to the FPV monitor through 5.8 GHz radio signal.

5.3.2 Hardware Design

In order to receive 2.4 GHz radio signals from the communication base, the quadcopter will carry a radio receiver module. The radio receiver module chosen for the quadcopter is FrSky D8R-II plus. However, since D8R-II can only output PWM signal and our flight controller only takes sbus as input protocol, we added a PWM-PPM-SBUS converter to enable a correct data flow. To translate the signal received from the radio transmitter to readable PWM signals for the ESC(Electronic Speed Control) and Motors, the quadcopter will have flight controller.

A flight controller is the most important component on the quadcopter. It is a microprocessor that controls the movements of the quadcopter. For this project, the flight controller is a Kakute F4 AIO V2 [17]. First of all, the Kakute F4 FC (Flight Controller) has the

F4 type of CPU, which has the processing speed of 168 MHz. This feature allows the quadcopter to be responsive and be able to react to any instruction within 0.6 seconds. Moreover, the Kakute FC has a built-in gyroscope chip and an accelerometer. The two components allow the quadcopter to fly smoothly and safely with the pre-installed program on the flight controller. The Kakute F4 FC is an open source flight controller, which means users are able to have access to the source code that defines the behavior of the quadcopter. This allows us to customize the flight controller and allow special commands such as landing and take off. Also, we will also be able to set the maximum speed of the quadcopter in this system to be 10 meters per second. In addition, the Kakute supports both FPV camera and FPV video data transmitter. This feature allows the quadcopter to carry an FPV camera and live stream video to the FPV monitor. However, the Kakute FC does not take PWM signals as input from the radio receiver. Thus, in order to convert the PWM signals generated by the 2.4 GHz radio receiver to a readable Sbus signal by the flight controller Kakute F4 FC, a signal converter module SBUS-PPM-PWM is necessary.

After the flight controller successfully receives the instructions from the communication base, it will start to generate PWM (Pulse Width Module) signals to the ESCs. An ESC is an electrical speed control that controls and regulates the speed of an electric motor. For the quadcopter in the gesture-controlled quadcopter system, the ESC will be Racerstar 30A V2. The ESC Racerstar 30A V2 is widely used with the flight controller Kakute F4. It allows for the smooth throttle response and silent operation of the quadcopter. Finally, the ESC will pass the ESC signals to each motor. In this way, the quadcopter could move according to the instruction from the communication base. The motors used for the quadcopter in this system is Racerstar 2207 which has maximum power of 482 watt that allows to quadcopter to be highly responsive.

In order to support all the electrical devices on the quadcopter, there will be a battery installed on it. The battery chosen for this quadcopter is XILO 1500mAh 4s 100c Lipo Battery. This battery is chosen because it is the cheapest possible Lipo battery that supports at least 10 minutes of flying time for the quadcopter in this system. To charge the battery, a battery charger EV-Peak E3 Falcore Edition 35W 3A LiPo Battery Balance Charger will be needed as well. This charger would allow the battery to be fully charged within 3 hours and is relatively cheap. To carry all the components on the quadcopter, a quadcopter frame will also be needed. The frame selected for this quadcopter is Martin II. This frame is a lightweight made with carbon fiber material which allows the quadcopter to be robust enough to not break if it falls from 3 meters high. Also, the diagonal of the frame is 220mm long, which meets our design requirement for the size of the quadcopter: less than a 25x25x25 cm^3 . More importantly, an FPV camera can be mounted on the Martian II frame. Thus, this frame suits the designed quadcopter very well.

For the video feedback, we are wiring our FPV camera, flight controller and video transmitter together. We wire the video out pin on our Foxeer Falkor to the video in pin on the flight controller. The raw video data is sent from the camera to the flight controller. The flight controller will compress and organize it into a sendable format. The processed video signal is sent to the video transmitter from video out to video pin on the video transmitter.

In sum, the hardware connection of the electronic components of the quadcopter in this project is shown below:

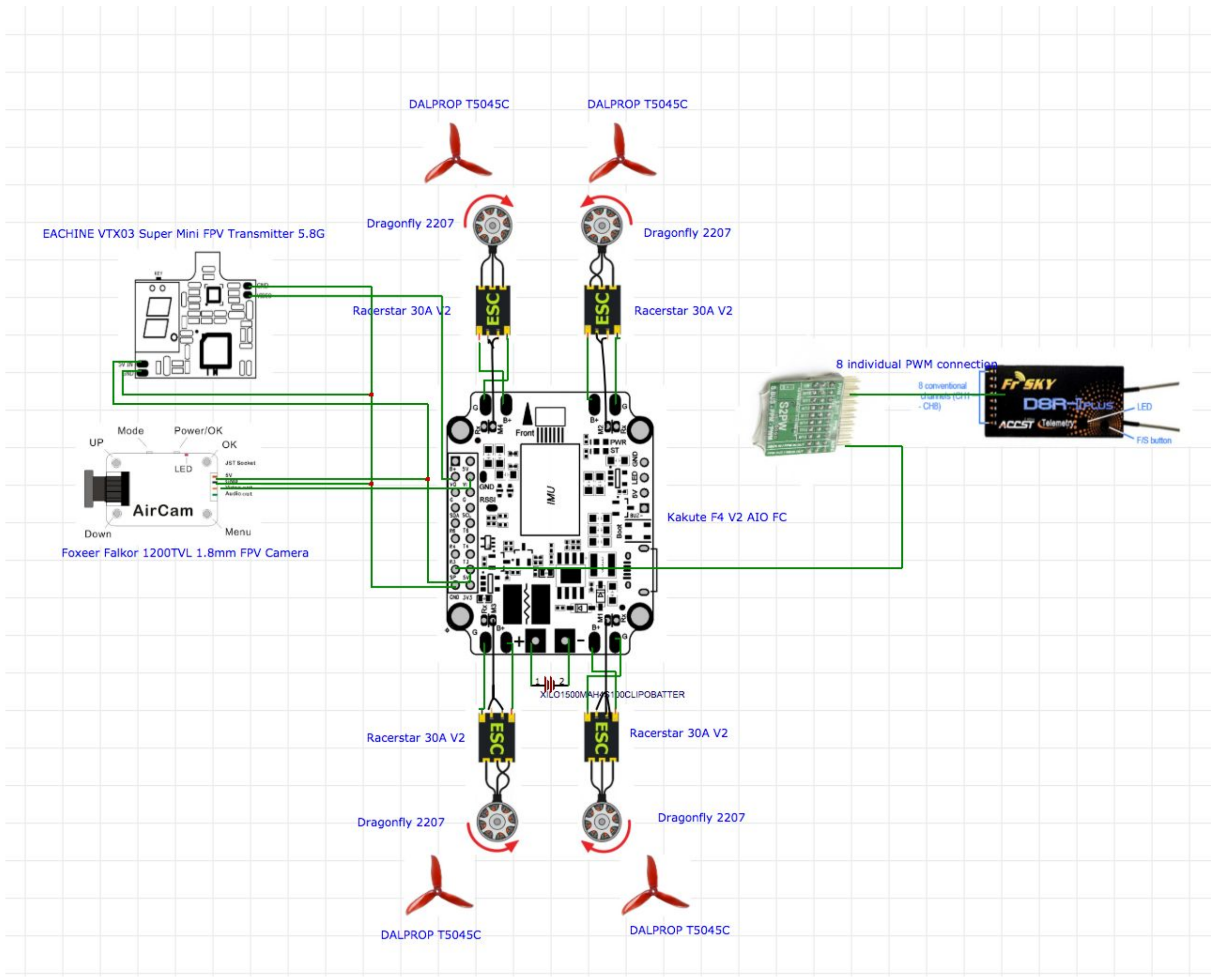


Figure 10. Schematic of the Quadcopter

5.3.3 Software Design

For the design of the quadcopter, there is very little software design involved. For example, components such as radio receiver, and PWM to Sbus signal converter only manipulates the taken data in hardware level. For components such as a flight controller and ESC, they only need to be configured before use by downloading the software drive provided by the companies for those products through a USB port. However, we can modify the source code of the flight controller and download it into it in order to limit the maximum speed of the quadcopter.

5.4 Data Flow

The data flow in this project can be rather convoluted due to the different communication protocols input and output by different communication components. Overall we can divide the data flow into three section: gesture controller to communication base, communication base to quadcopter and quadcopter to communication base.

The data flow originated from the sensor mounted on the gesture controller. The microprocessor will collect the data and convert it into a string that can either be the current motion data or special command such as land or take off. The string is then sent to the communication via a 433Mhz radio signal.

For the second part of the communication, the base will take in the string received from the 433Mhz module. The base will follow the internal finite state machine to interact with the input string and transmit the flight command to the radio transmitter in the form of PPM signal. The transmitter would convert the information in the PPM signal into flight command and sent it to the quadcopter through 2.4GHz radio. On the quadcopter the information is received and decoded into eight PWM signals. However, for the requirement for the flight controller on quadcopter the eight PWM signals are changed into S-Bus protocol. The flight controller interpret the command and change the speed and direction of its motor.

The last communication is simple. The video is transmitted back to the monitor on base in 5.8Ghz radio signal by the video transmitter mounted on quadcopter.

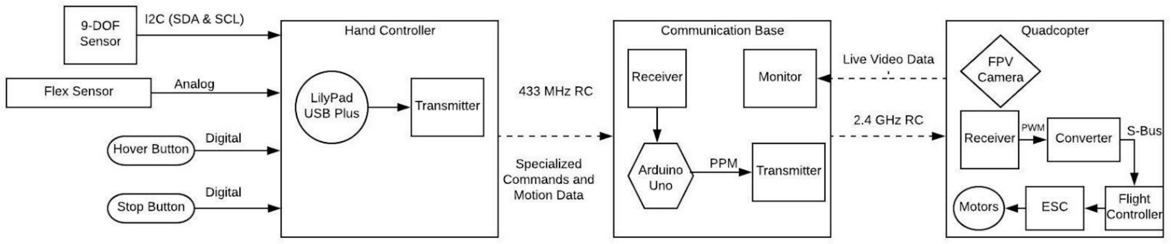


Figure 11. Diagram of the Data Flow of the System

6. Final Design and Implementation

6.1 Hand Controller

6.1.1 General Design

The general design for the hand controller has not had any severe changes from the original design. It is still broken up into three distinct tasks reading the hand's position and specialized commands. The hand controller still converts the information but instead of into a string it does it into a series of unsigned integers which will be discussed further in the software design. Finally, the third task is to send the information to the communication base via the radio communication modules. The final product uses conductive thread to connect the components with the intention of the hand controller being able to be used on both hands. The hardware does not impede the motion of the hand when reading the positional data.

6.1.2 Gesture Design

The gestures are still based on the Aura toy produced by KD Interactive USA [2]. The figure below shows the gestures we adapted to our system.

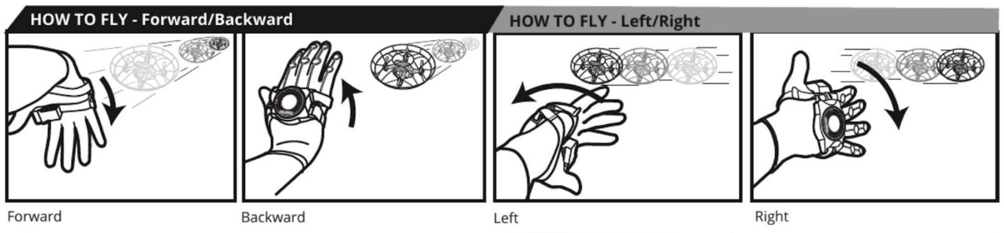


Figure 12. Hand Gestures Instruction for Aura

The previous design showed that the throttle would be controlled by pressing a button and repeating the forward and backward commands. We originally decided that the throttle would have been controlled by the vertical movement of the hand. We decided against the original design because we had difficulty getting a steady signal from the gyroscope from the 9-DOF board. To solve this problem we decided to utilize the flex sensor since it's the only function is to initialize the take-off function. While the hand controller is enabled and reading positional data if the flex sensor is engaged the throttle will increase until the flex sensor is unengaged or the hand is flipped and it will decrease the throttle. This will increase the range which the quadcopter can go without having to engage the hover special command.

6.1.3 Hardware Design

The previous design included a schematic of the hardware design that was implemented. It is included below:

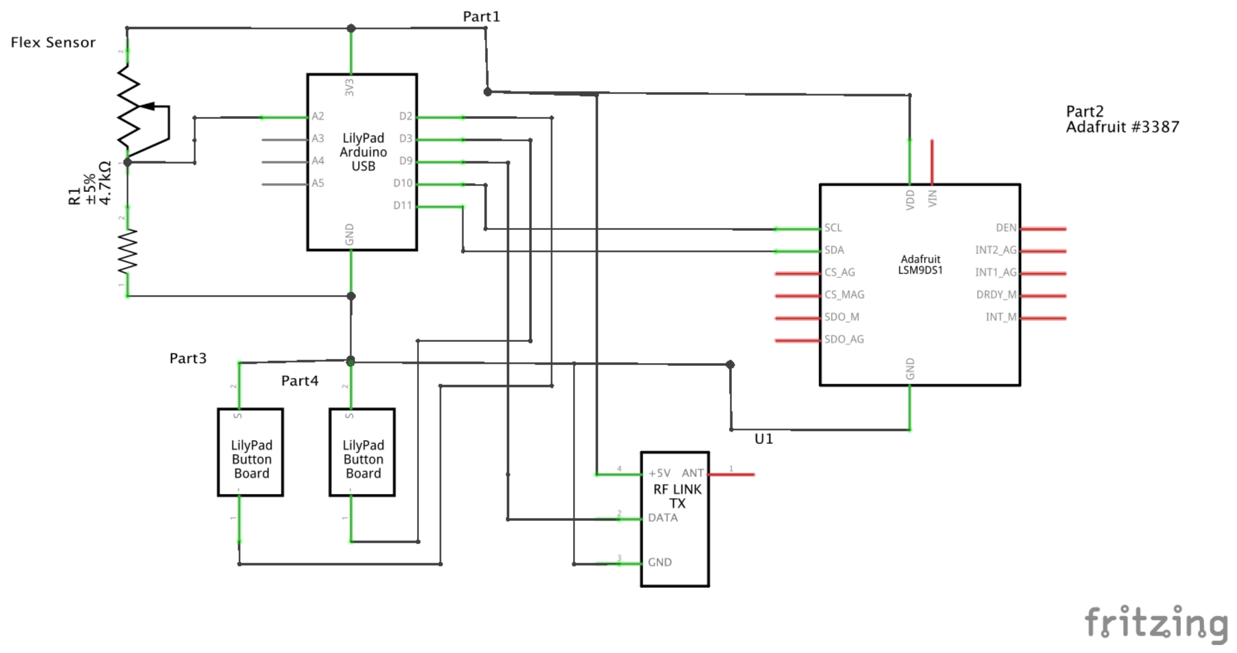


Figure 13. Schematic of the Hand Controller

The physical system is designed exactly as the schematic is laid out. There are two LilyPad buttons, an analog flex sensor (bend resistor), a 433MHz Radio Transmitter, the 9-DOF sensor, and the LilyPad USB Plus. This is sewn onto a piece of stiff felt fabric with conductive thread. The conductive thread is covered in craft fabric paint to insulate the wires. This is important because skin contact between the threads is conductive enough, over short distances, to pull a pin to ground. Below the implementation of the system can be seen.

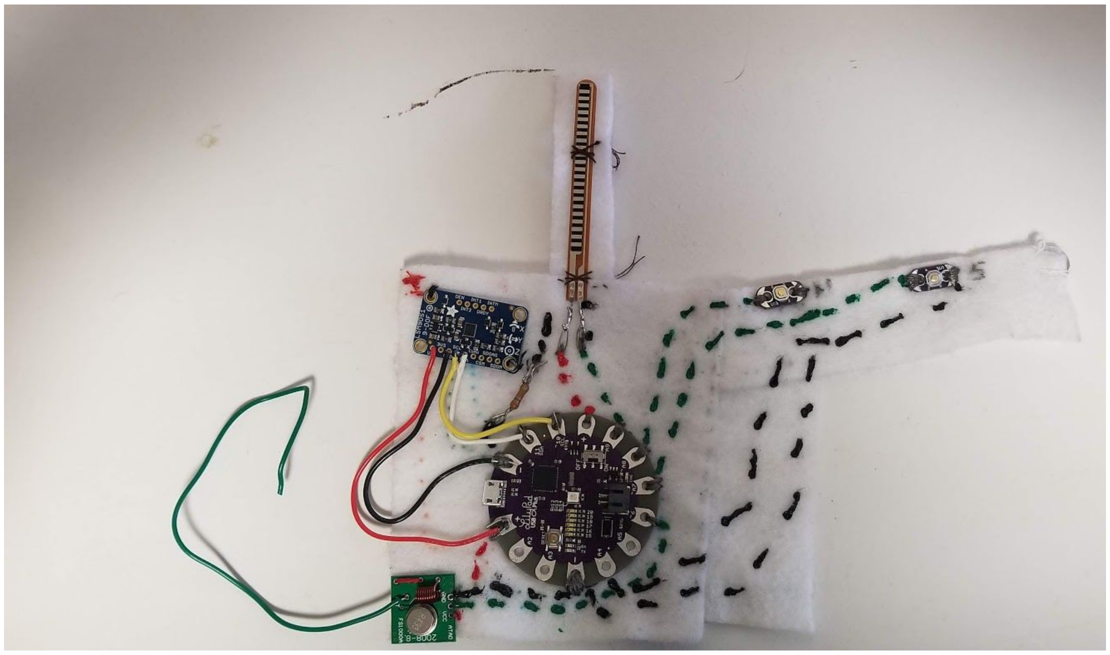


Figure 14. Hardware Implementation of the Hand Controller

6.1.4 Software Design

The biggest changes from the preliminary design wherein the software design. The system still uses Arduino because of the libraries that could be downloaded to use with the components purchased. The software design is split into three major components the first being the specialized commands, the second is the positional data reading and the third is the radio communication.

The specialized commands are a series of software-based interrupts. Though this is not the best way to implement this system because it can cause delay when a button is pressed, read, and processed. This impact and ways of solving it will be discussed later in the results section. The three specialized commands are take off, hover, and stop as mentioned before in the preliminary proposed design. The state diagram below shows how the system is implemented:

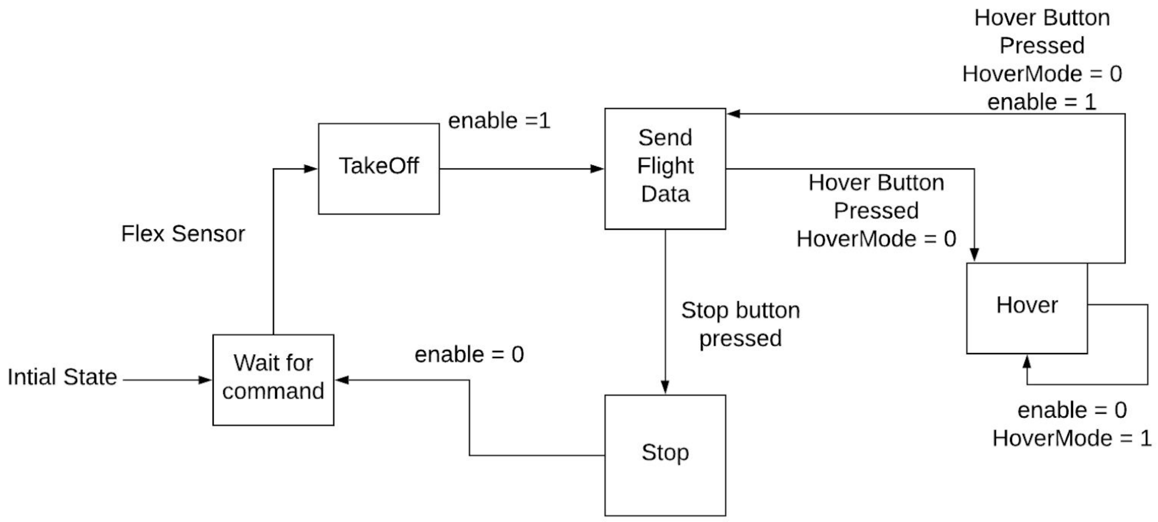
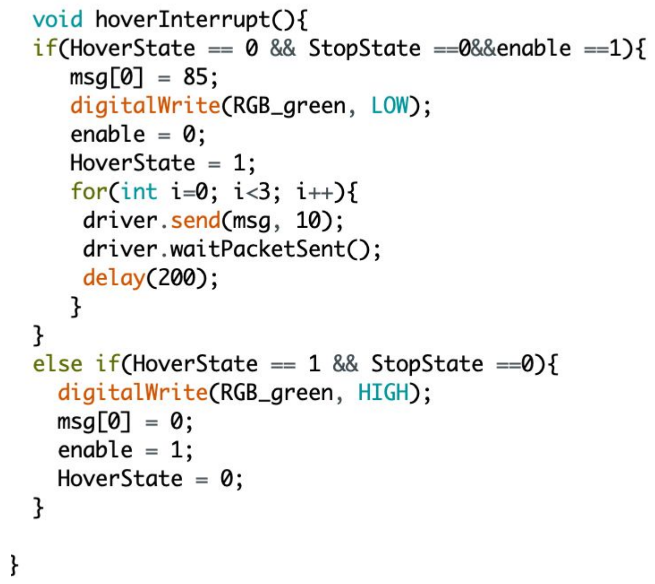


Figure 15. State Diagram for the Special Commands

The reason that Stop and Hover interrupts have to be implemented with software-based interrupts is that the LilyPad USB Plus is a modified ATMega32 chip and the only interrupt pins available are designated SDA and SCL Pins. Ways to solve this problem will be discussed later in the results and discussion section of the paper. The functions designed to be the interrupt are shown below.

```
void stopInterrupt(){

     msg[0] = 170;
     digitalWrite(RGB_green, LOW);
     enable = 0;
     StopState = 1;
     HoverState = 0;

      for(int i=0; i<3; i++){
      driver.send(msg, 10);
      driver.waitPacketSent();
      delay(200);
     }
     msg[0] = 50;

}
```

Figure 16. Stop Interrupt Function

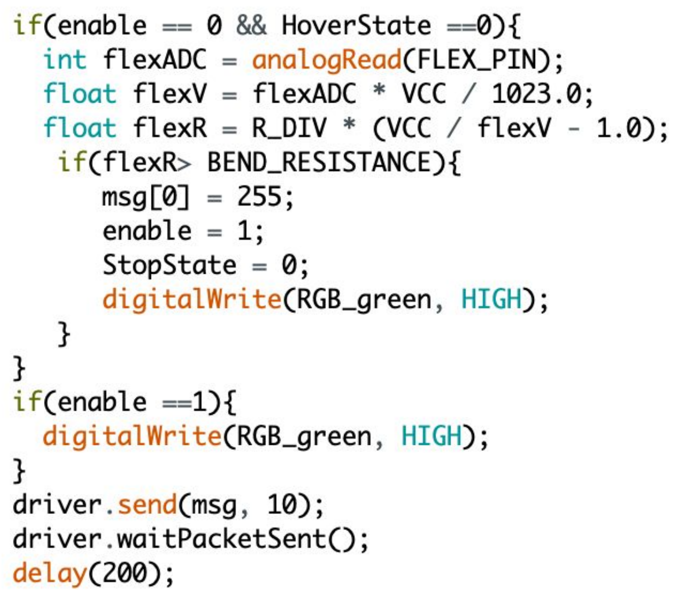


```
void hoverInterrupt(){
if(HoverState == 0 && StopState ==0&&enable ==1){
   msg[0] = 85;
   digitalWrite(RGB_green, LOW);
   enable = 0;
   HoverState = 1;
   for(int i=0; i<3; i++){
    driver.send(msg, 10);
    driver.waitPacketSent();
    delay(200);
   }
}
else if(HoverState == 1 && StopState ==0){
  digitalWrite(RGB_green, HIGH);
  msg[0] = 0;
  enable = 1;
  HoverState = 0;
}

}
```

Figure 17. Hover Interrupt Function

As shown both the stop and hover functions edit the enable variable this enable variable controls whether or not the system reads the positional data. Both the Stop State and Hover State have variables that show whether or not that is the current state. The stop function can be enacted at any point even if the system is in the hover state. The full hand controller code can be seen in Appendix 2. In that code, you can see every couple of lines an if statement reads the stop and hover buttons and if they're pressed they jump into the stop or hover functions.

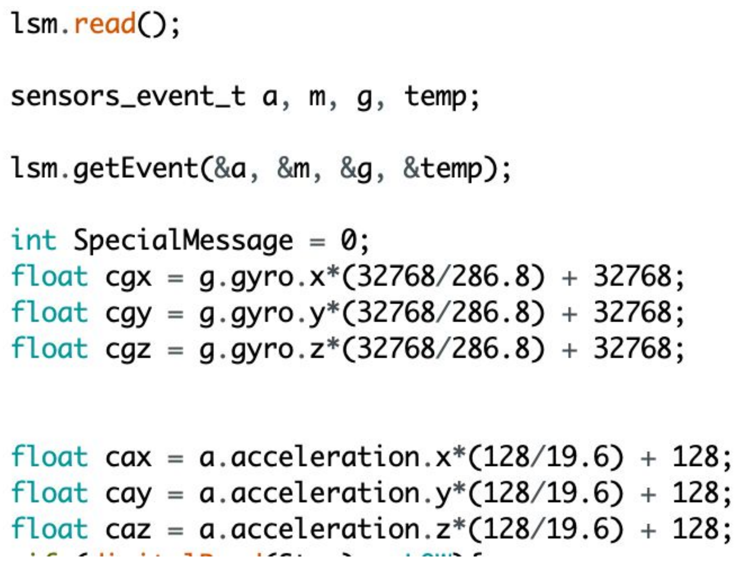
The take-off function is a bit different than the other two special commands. It is not the same because it cannot be enacted at any point it is only a viable option when the system is not enabled and the HoverState is also low. Both of these things mean that the system landed and ready for taking off. The flex sensor wouldn't be enacted until the resistance is greater than a certain BEND_RESISTANCE. Currently, it is set to activate when the bend is greater than 90 degrees. Below is the code that uses the flex sensor to indicate the take-off function.



```
if(enable == 0 && HoverState ==0){
  int flexADC = analogRead(FLEX_PIN);
  float flexV = flexADC * VCC / 1023.0;
  float flexR = R_DIV * (VCC / flexV - 1.0);
   if(flexR> BEND_RESISTANCE){
      msg[0] = 255;
      enable = 1;
      StopState = 0;
      digitalWrite(RGB_green, HIGH);
   }
}
if(enable ==1){
  digitalWrite(RGB_green, HIGH);
}
driver.send(msg, 10);
driver.waitPacketSent();
delay(200);
```

Figure 18. Take Off Function

The second component of the software design is the ability to read the positional data of the hand and convert it to data that can be sent. Below shows the code that does that.

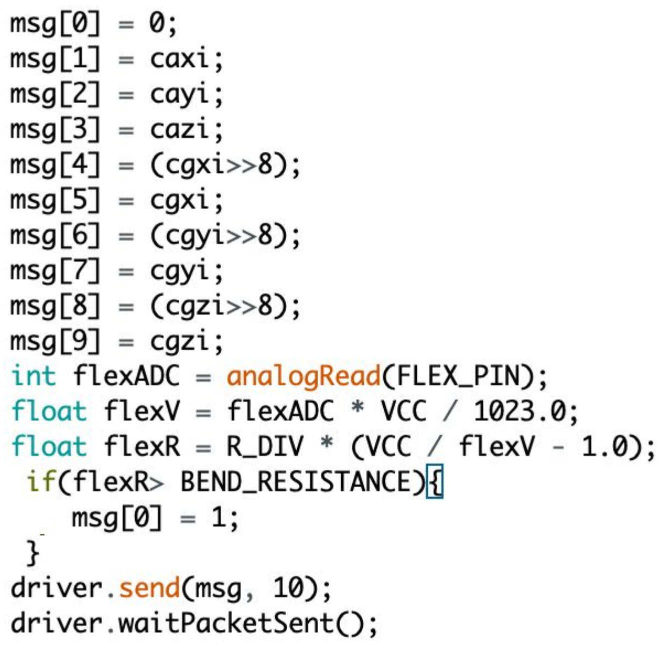


```
lsm.read();

sensors_event_t a, m, g, temp;

lsm.getEvent(&a, &m, &g, &temp);

int SpecialMessage = 0;
float cgx = g.gyro.x*(32768/286.8) + 32768;
float cgy = g.gyro.y*(32768/286.8) + 32768;
float cgz = g.gyro.z*(32768/286.8) + 32768;


float cax = a.acceleration.x*(128/19.6) + 128;
float cay = a.acceleration.y*(128/19.6) + 128;
float caz = a.acceleration.z*(128/19.6) + 128;
```

Figure 19. Positional Data Code used to Read the Hand's Position

The library that is used to read the data from the 9-DOF board is the LSM9DS1 Library provided by Adafruit. It has built-in functions that allow for easy access of the data on the I2C bus that the 9-DOF board uses. The library has the ability to access all the information needed to be able to tell the position of the hand. As can be seen in the code above both the gyroscope and the accelerometer are adjusted by different values. For both accelerometer and the gyroscope, the number that is dividing 32768 and 128 respectively is the max values we were able to achieve by movements of the hand with the minimum being the negative version of the number. For the communication the values they need to be 8-bit unsigned integers. The range of the gyroscope needed to be much larger than that to encapsulate a large range of data. Since we needed more space for the gyroscope data we adjusted it for 16-bit values, the 32768 is half the bit value that can be represented by 16 bits. Since the accelerometer can be represented by a smaller range it is only represented by 8 bits.

The final major component of the hand controller software is the communication between the hand controller and the communication base. This achieved by the 433MHz radio communication we used the Radiohead library to facilitate the communication. The code used for communication is seen below.



```
msg[0] = 0;
msg[1] = caxi;
msg[2] = cayi;
msg[3] = cazi;
msg[4] = (cgxi>>8);
msg[5] = cgxi;
msg[6] = (cgyi>>8);
msg[7] = cgyi;
msg[8] = (cgzi>>8);
msg[9] = cgzi;
int flexADC = analogRead(FLEX_PIN);
float flexV = flexADC * VCC / 1023.0;
float flexR = R_DIV * (VCC / flexV - 1.0);
 if(flexR> BEND_RESISTANCE){
    msg[0] = 1;
 }
driver.send(msg, 10);
driver.waitPacketSent();
```

Figure 20. Code Used to Send a Message to the Communication Base

By using the RadioHead library the function send requires a list of unsigned 8-bit integers and the length of the array. The system we made needed 10 bytes of data to be sent to the communication base. We decided that the first index of the array would hold the special command values, 0 being no special command, 85 being hover, 170 being stop, 255 being take-off, 50 being the disable command, and 1 being the throttle read. They're all allowed to be represented by one index because the value can range from 0-255 distinct values. We found that even though the communication isn't the most stable it does not change the values of the messages received. The next three indices are the unsigned 8-bit integers that represent the accelerometer data. The next six indices represent the gyroscope data the concurrent indices represent the upper 8 bits and the lower 8 bits of each x, y, and z. The library also has a function that waits for the system to send the package of data before it would have the ability to send another package of data.

6.2 Communication Base

6.2.1 General Design

The overall design for the communication base had not changed much from the original design either. The functionality of communication base remains a two one-way data processing. One that receive controller data from the hand controller, decode and transmit the corresponding flight command to the quadcopter, whereas the other one receive and display the video signal on the monitor. On the hardware level, not much are changed beside a 5.8GHz radio receiver is added to receive the 5.8Ghz video signal, a component missed during the original design. More changes are made in terms of software. As the finite state machine for special command is implemented in the hand controller, the communication base would only need to cover the special command handling, gesture interpretation and conversion of user intention to flight data. The changes will be discussed more detaily in following subsection session.

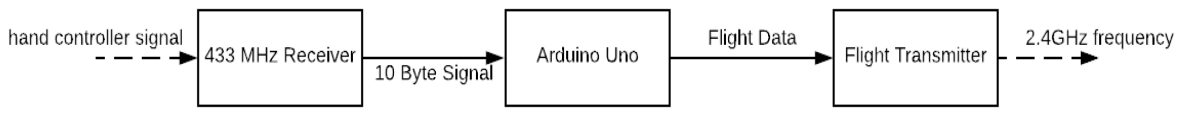


Figure 21. Data Flow for User-Input Controls to Flight Data

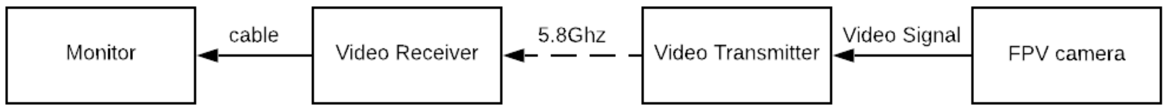


Figure 22. Data Flow for Video Stream

6.2.2 Hardware Design

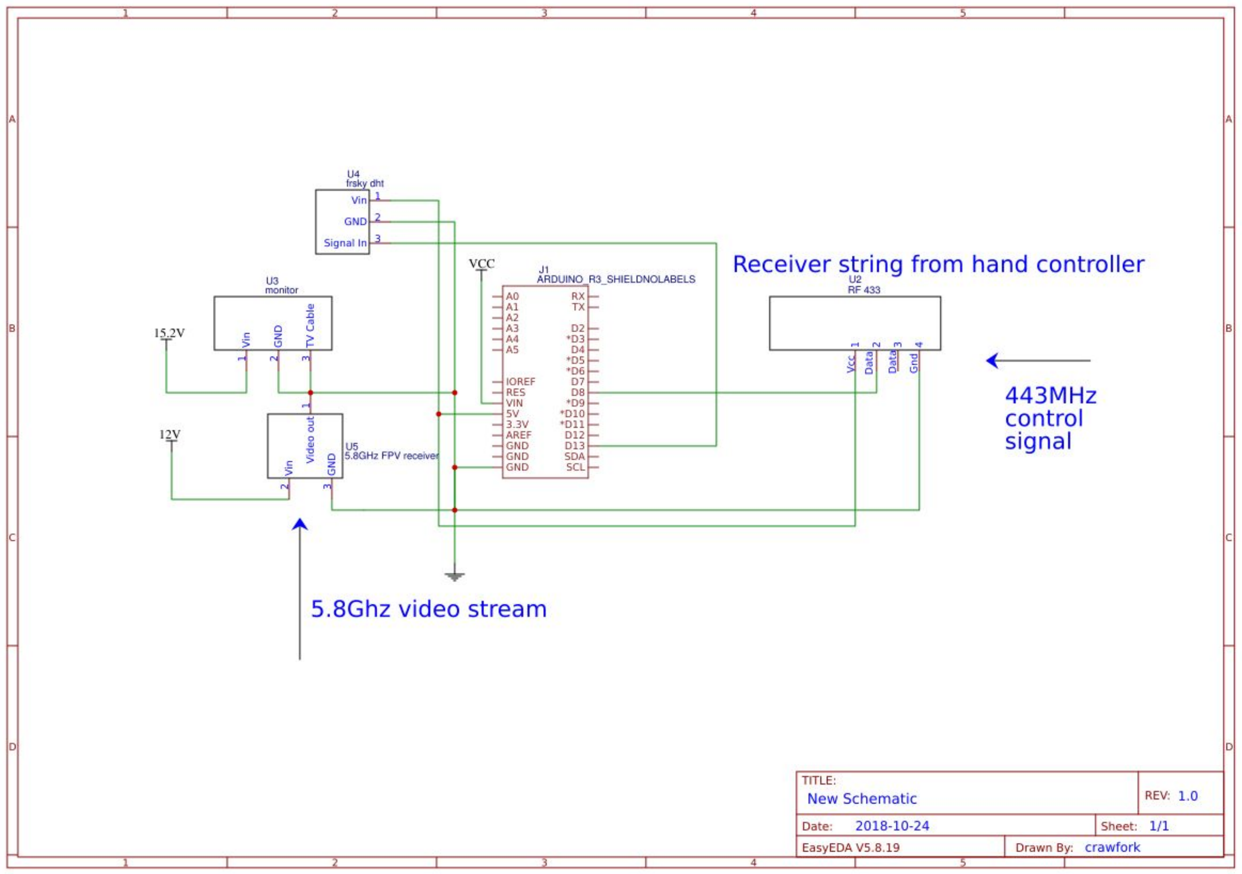


Figure 23. Final design Diagram for Communication Base

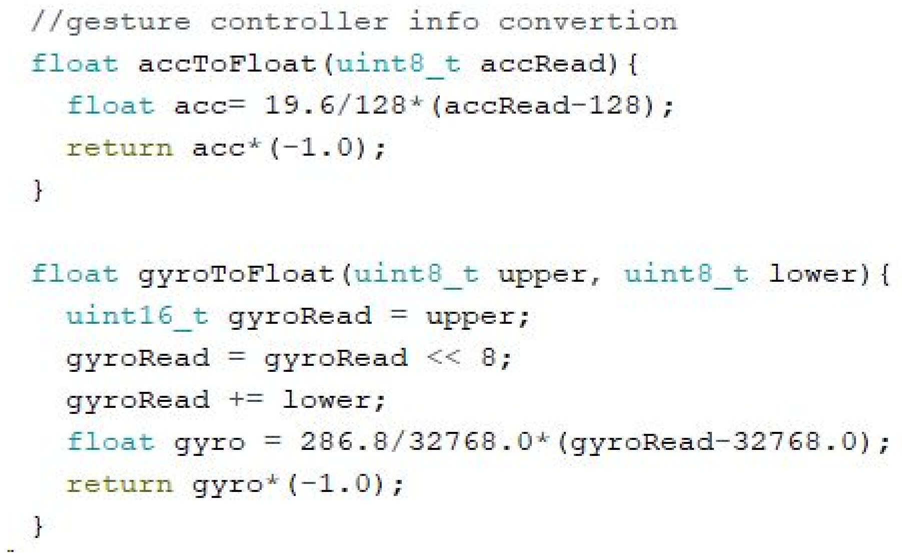
As we mentioned in the previously, the communication has not changed much beside a 5.8 FPV video receiver is added. There is a 433MHz radio receiver to acquire data from hand controller, an Arduino Uno to interpret user intention, a FrSky DHT 2.4 GHz radio transmitter to send flight data to drone, and 5.8 Ghz FPV receiver and monitor for video streaming. The wiring is simple as there is only five components involved and each components only needs three wiring connection. However, currently the communication is still implemented on a breadboard and not in a condition to be compacted into a portable device. We will discuss the reason behind this in the conclusion section.

6.2.3 Software Design

The software design had been modified to adjust to the change in the hand controller and hardware limitations. As the finite state machine is implemented on the hand controller, the base does not need to memorize which state the quadcopter is in anymore. Thus the base can be more focused on its main tasks: data conversion and user intention interpretation.

Data conversion is necessary as we have different components that provide/require data in different format. The 433MHz radio receiver will return an array of byte for every transmission whereas FrSky DHT transmitter need PPM signal, a special data format that is used to compress data from multiple channel to one channel.

The part fetch data from 433MHz receiver is simple. There is a library “radiohead” that provide the function that return the received byte array from the radio receiver. And as we had seen in hand controller section, the data is formatted in 10 byte per transmitter where the first byte is an indicator for special command, the next three byte for the three axis reading from accelerometer, and the last six byte for the three axis reading from gyroscope as the range of possible data is wider and need more precision. Thus the conversion from byte back to the actual data is just the reverse of the conversion in hand controller. The implementation is listed below:



```
//gesture controller info convertion
float accToFloat(uint8_t accRead){
  float acc= 19.6/128*(accRead-128);
  return acc*(-1.0);
}

float gyroToFloat(uint8_t upper, uint8_t lower){
  uint16_t gyroRead = upper;
  gyroRead = gyroRead << 8;
  gyroRead += lower;
  float gyro = 286.8/32768.0*(gyroRead-32768.0);
  return gyro*(-1.0);
}
```

Figure 24. Byte to Float Conversion Code

The PPM signal generation is a bit more complex. As we had mentioned before, PPM signal combines multiple channel of data, in our case, eight, into one channel. It compress the data by having the number of channels plus 1 pulses in each period. The length of the pulse represent the corresponding channel value. The longest extra pulse marks the end of current period and the beginning of the next period.

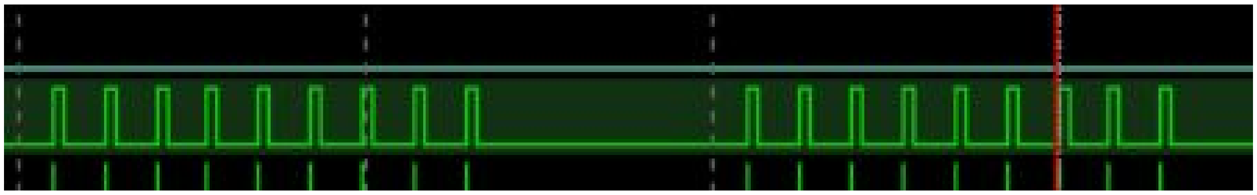
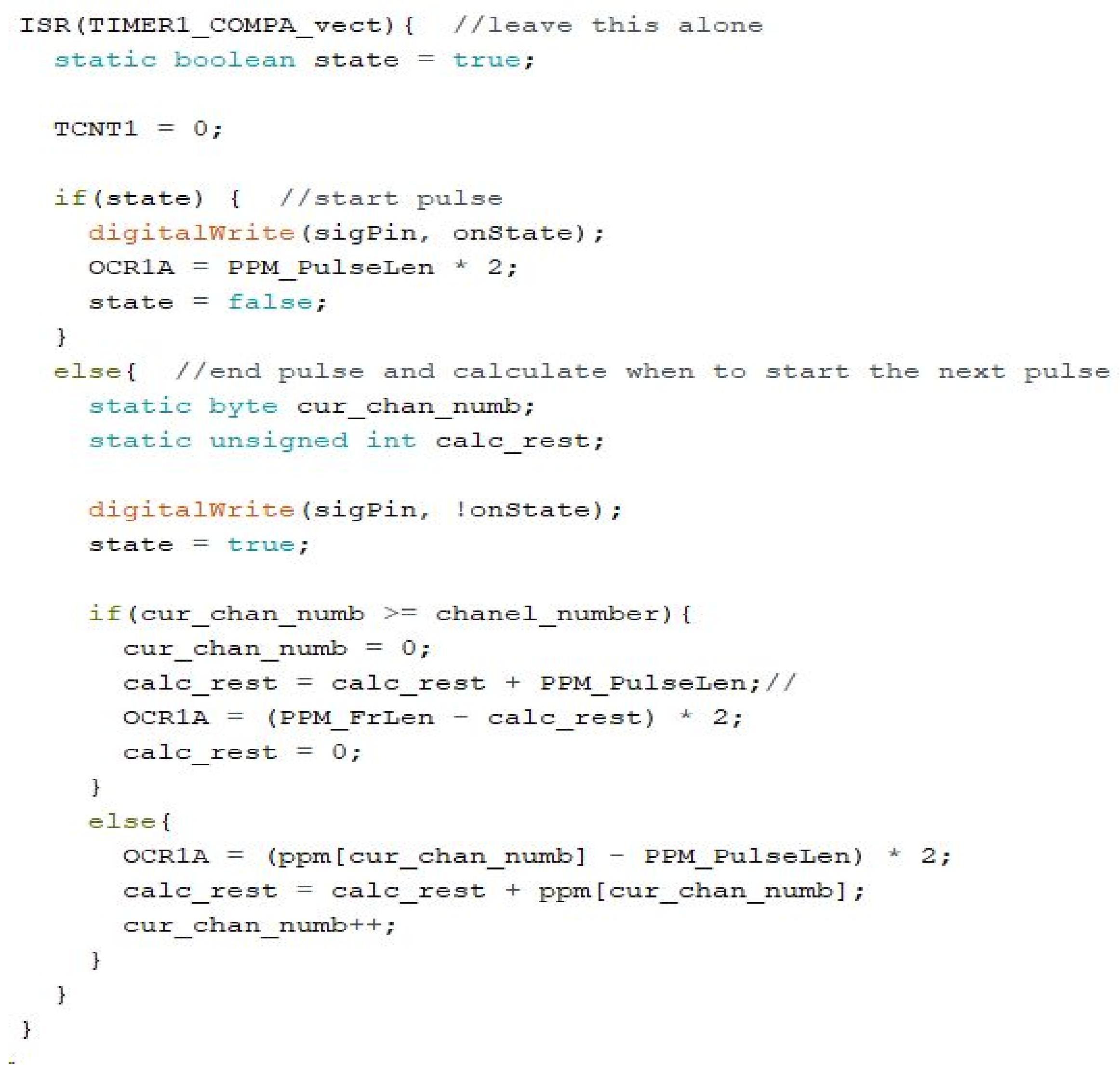


Figure 25. Generated PPM Example from Base Measured by Logic Analyzer

After knowing the structure of the PPM signal, the signal generation becomes trivial. The value for each channel is stored in an array. A timer interrupt function is used to periodically visit the array and create the corresponding pulses. This implementation allow us to separate the PPM generation function from the main loop to achieve a cleaner design. The interrupt function is shown below:



```
ISR(TIMER1_COMPA_vect){  //leave this alone
  static boolean state = true;

  TCNT1 = 0;

  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PPM_PulseLen;//
      OCR1A = (PPM_FrLen - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}
```

Figure 26. Timer Interrupt Function for PPM Generation

An interesting side note for the data conversion is that both radiohead library and the PPM generation interrupt function use the build-in timer in the microcontroller. By default both function uses timer1, which will create a failure when compiling the code. While changing RadioHead library to use timer2 is easy, it is something to notice when choosing microcontroller for communication base as some microcontrollers do not have 2 build-in timer.

Once we setup the infrastructure for the communication base, we can start to use the data to understand the user's intention and translate the intention to the correct flight data. Once receiving a new array of data, the base would first check if the given command is a special command or not. If it is a special command, the corresponding function will be called. If it is not a special command, the base would attempt to interpret the gesture with the data from accelerometer and gyroscope. Thus we can divide this part to two smaller components: function mapping for special commands and gesture interpretation.

As a reminder, for now there are four special commands: take off, hover, land and stop, which is a kill switch that shut down the quadcopter immediately. We plan to have one function for each special command, but we do not have a command for land yet. The reason behind this will be discussed later in detail. Also we have not tested out the correct value for the special command so here we are only providing the general ideas for the functions that corresponding to the special commands.

To successfully implement the kill switch function, we need to introduce the arm mode on the quadcopter first. Arm mode is a safety feature to protect both the quadcopter itself and the surrounding. When a quadcopter is not armed, all the motor is automatically shutted down. So we can simply set a channel to present the arm mode and when switch the arm channel to off to shut down the quadcopter. A button with the same function is also implement to ensure minimal delay. However this also means introducing a state in the PPM generation interrupt function so the arm channel does not automatically turn back on.

We also need to use our knowledge of arm mode in take off function. Take off function will only be triggered when the quadcopter is landed. This means that the arm mode is going from off to on. However, the arm mode will not be turned on unless the arm mode is off for a few seconds. This implementation is used to prevent the noise in the controller to accidentally move the drone. But this also means that we need to set the arm mode to off first before we turn on the arm mode and put in the correct throttle value.

For the hover function, we will just have the quadcopter to retain a certain value of throttle, to make the quadcopter hover in the air. However we do not have the exact number for the quadcopter yet as we do not have a chance for testing. In the future we can add in a GPS module for a stabler hover.

Last but not least, as I mentioned above, we have not implement the landing function as there is not a component on the quadcopter that can measure it’s distance to the ground. To implement this function we also need the quadcopter be able to send feedback to the communication base. This is definitely a useful feature for the future work.

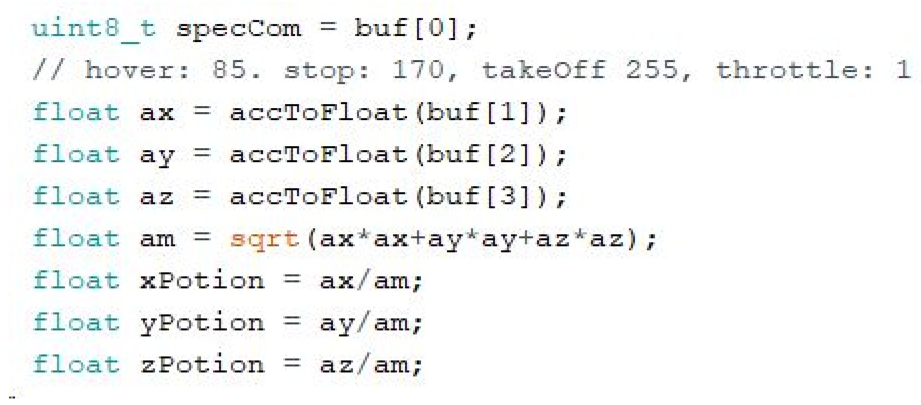
The other part of user intention interpretation is connecting the user gesture to the corresponding flight command. We have a naive algorithm that uses the data generated by accelerometer and gyroscope to estimate the position and orientation of the hand. Again due to connection issues we do not have a chance to test the effectiveness of the algorithm in real life.

However, before we can discuss the algorithm, we need a short introduction to what data do accelerometer and gyroscope provide. Accelerometer measures the net acceleration, or equivalently, force, on each axis. Provided that the hand controller is at rest, we know the only force that is exerted on the accelerometer is gravity. Thus we know the magnitude of the accelerometer reading on the three axis is $9.8 m/s^2$. By calculating the ratio of acceleration on each axis to the total magnitude, we can find out the angle on each axis, and thus the current position and orientation of the hand.

Gyroscope, on the other hand, measures the rotation speed on each axis. Thus given an initial position and the time interval between each probing, we can keep track of the current angle on each axis by integration.

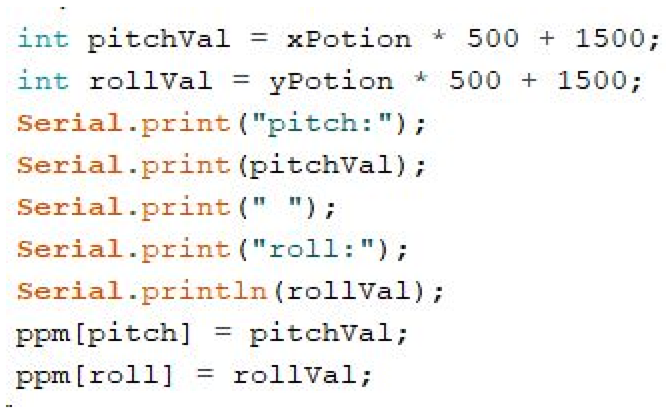
If we combined the position measured by the accelerometer and gyroscope, we can have a relatively accurate reading for the position of the hand. If the hand is at rest, the reading for x,y direction is close to 0, where as when value on x, y axis changed near to 1 or -1 means the hand is tilted in that direction, which can be converted to a corresponding value in pitch or roll.

However currently the gyroscope have a big noize in its reading and we cannot find out a way to clean the data so we are currently only relying on the accelerometer to estimate the hand position. Below is our naive algorithm:



```
uint8_t specCom = buf[0];
// hover: 85. stop: 170, takeOff 255, throttle: 1
float ax = accToFloat(buf[1]);
float ay = accToFloat(buf[2]);
float az = accToFloat(buf[3]);
float am = sqrt(ax*ax+ay*ay+az*az);
float xPotion = ax/am;
float yPotion = ay/am;
float zPotion = az/am;
```

Figure 27 Data collection and position calculation



```
int pitchVal = xPotion * 500 + 1500;
int rollVal = yPotion * 500 + 1500;
Serial.print("pitch:");
Serial.print(pitchVal);
Serial.print(" ");
Serial.print("roll:");
Serial.println(rollVal);
ppm[pitch] = pitchVal;
ppm[roll] = rollVal;
```

Figure 28. Position to flight command

6.3 Quadcopter

6.3.1 General Design

There are two general design goals for the quadcopter subsystem. First, the quadcopter has to fly according to the signals received from the communication base. Second, the quadcopter has to live stream video data to the FPV monitor on the communication base. Since the design goals remain unchanged, the two datapaths are the same from the preliminary design: one for quadcopter control, the other for live video streaming.

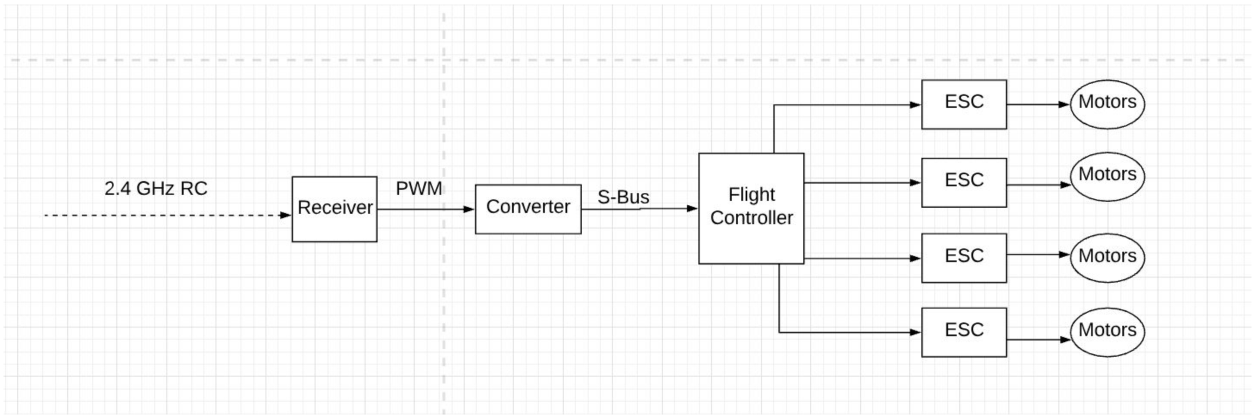


Figure 29. The movement control datapath

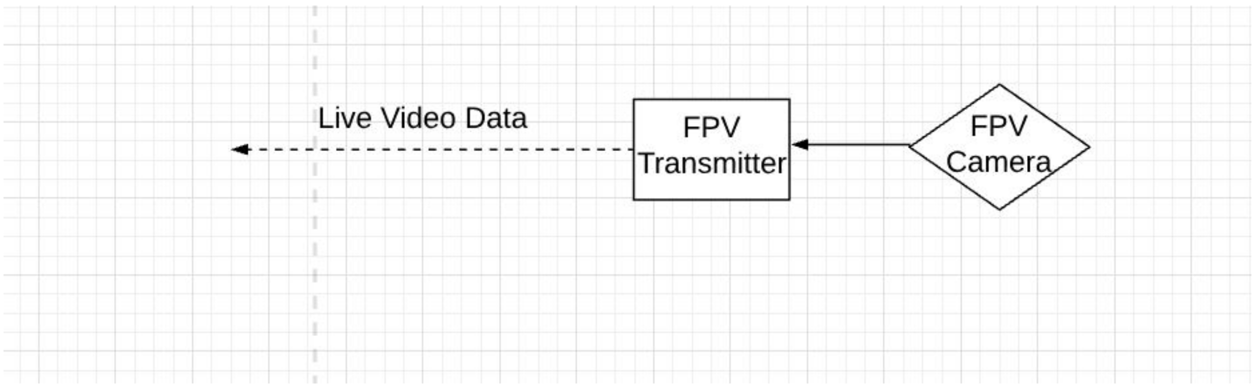


Figure 30. The Live Video Streaming Datapath

6.3.2 Hardware Design

For the hardware implementation, there is no major change from the preliminary design has been made. All the components needed remain to be the same. Below is the schematic of the quadcopter, the component list and the final hardware implementation.

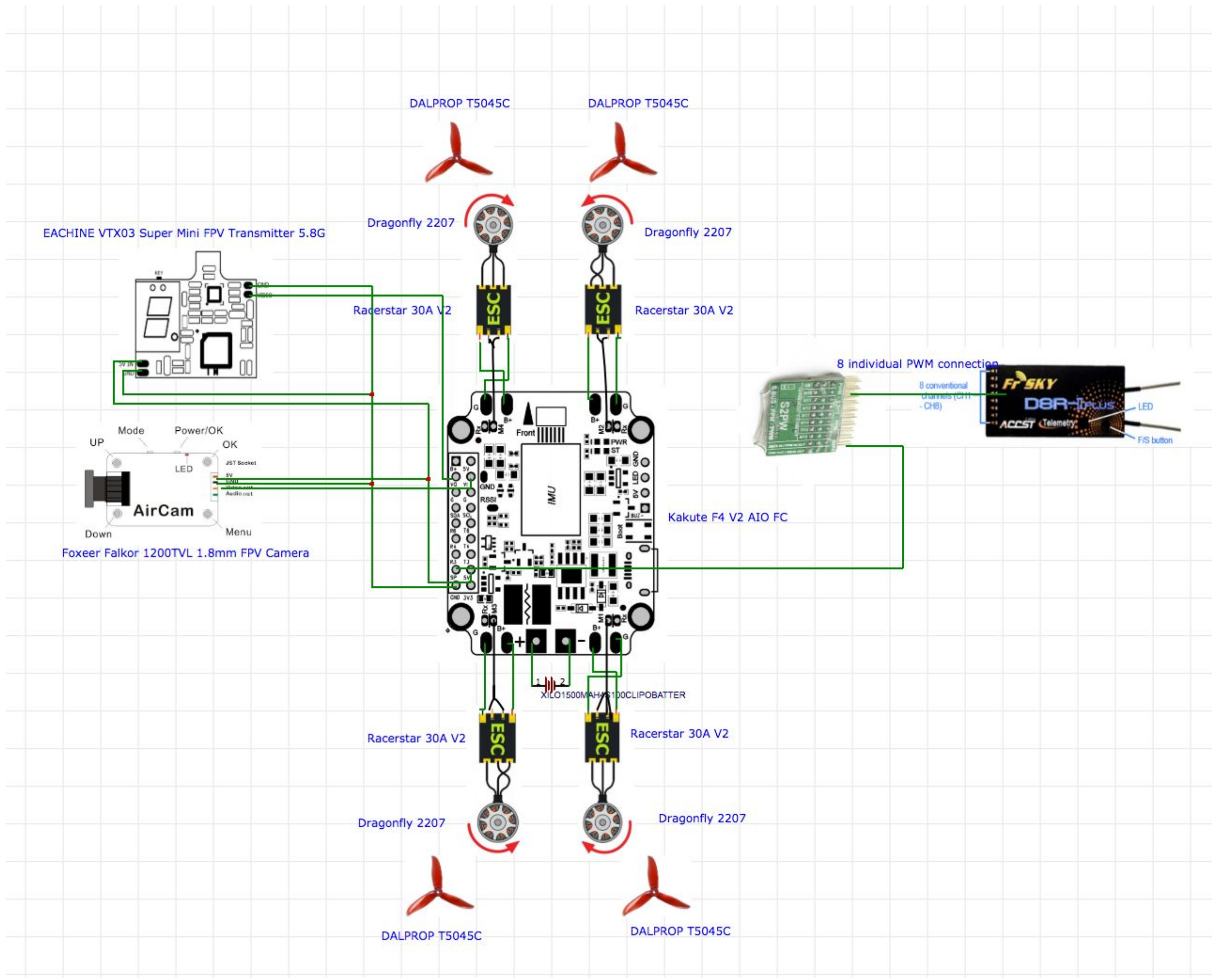


Figure 31. Quadcopter Hardware Schematic

Component	Chosen Model
FPV Camera	Foxeer Falkor 1200TVL 1.8mm FPV Camera - Limited Edition White

FPV Transmitter	EACHINE VTX03 Super Mini FPV Transmitter 5.8G 72CH 0/25mW/50mw/200mW Switchable Transmission
Frame	Martian II
Flight Controller	Kakute F4 V2 AIO FC
Motors	Racerstar 2207
ESC	Racerstar 30A V2
Propeller	DALPROP T5045C Cyclone
Battery	XILO 1500mAh 4s 100c Lipo Batter
Signal Converter	SBUS-PPM-PWM(S2PW)
Radio Receiver	FrSky D8R-II PLUS 2.4GHz 8CH Receiver with Telemetery

Table 7. Components List for the Quadcopter

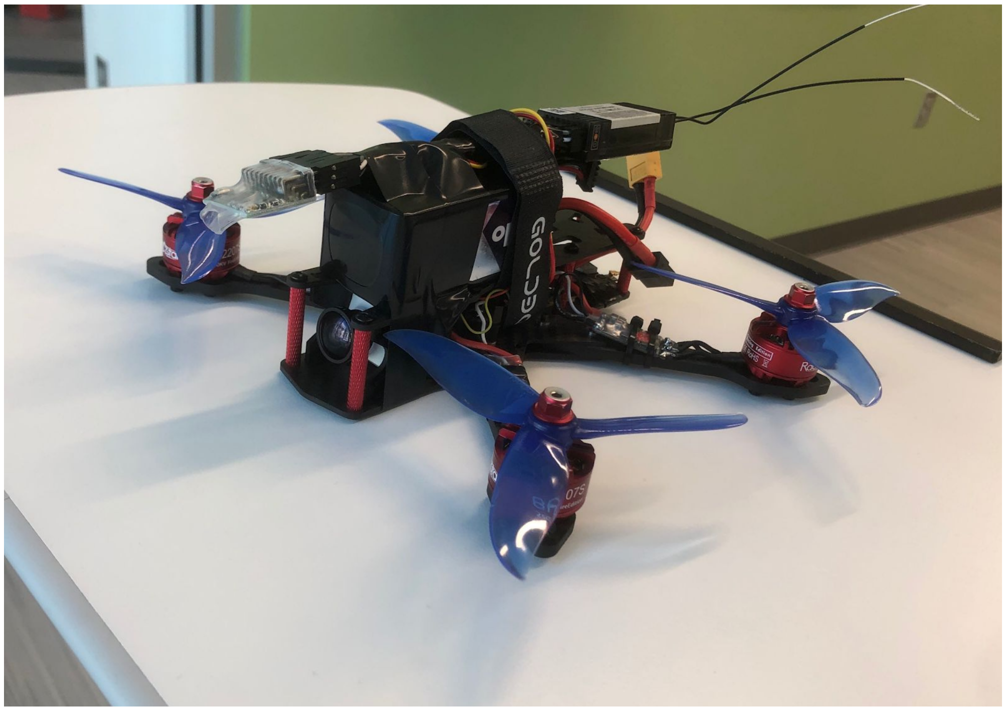


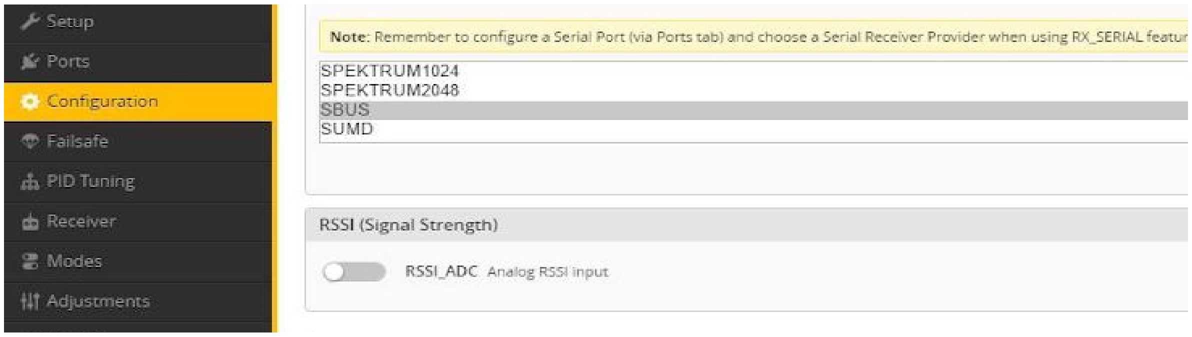
Figure 32. Final Hardware Implementation for the Quadcopter

6.3.3 Software Design

As mentioned in the preliminary design, there is very little software design involved in the quadcopter subsystem. However, there are some important configurations needed to be made for flight controller for the quadcopter to be properly manipulated by the communication base. A

USB cable can be used to make the physical connect between a computer with Betaflight Configurator[19] installed and a flight controller.

In order to notify the flight controller that a SBUB signal is used in this system to carry the control signal, in the Betaflight configurator, the receiver type has to be selected as SBUS in the Configuration page.



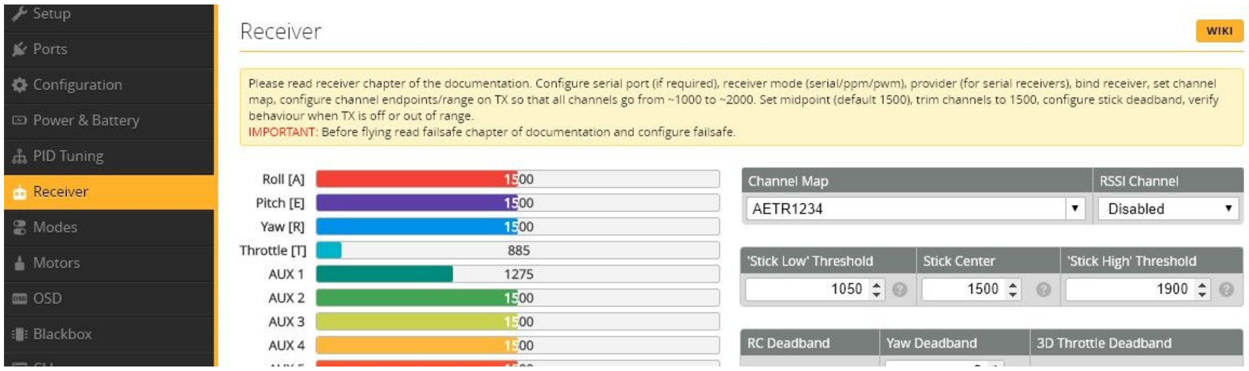
Note: Remember to configure a Serial Port (via Ports tab) and choose a Serial Receiver Provider when using RX_SERIAL featur

SPEKTRUM1024
SPEKTRUM2048
SBUS
SUMD

Figure 33. Receiver Configuration

The second configuration is for the receiver channel mapping. In the Receiver page of the configurator, there are receiver channels with values on the left side of the user interface, and configuration options on the right side of the user interface. Because the receiver of this quadcopter subsystem is a 8-channel receiver, it only receivers value for 8 channels, which are: Roll, Pitch, Yaw, Throttle, AUX1, AUX2, AUX3, and AUX4. Each channel has valid value varying from 1000 to 2000. The Roll, Pitch, Yaw, Throttle channel are used to specify the basic movements of the quadcopter, and the other AUX channels are used for different modes of the quadcopter. For current design, the Channel Map is set as AETR1234, which means the first four channels on the left side are mapped to control the Poll, Pitch, Throttle, Yaw values. On the right side of this interface, there are options can be set such as “Stick Low” and “Stick High”

Threads. Those configurations can be used to set maximum speed of the quadcopter.



Receiver

Please read receiver chapter of the documentation. Configure serial port (if required), receiver mode (serial/ppm/pwm), provider (for serial receivers), bind receiver, set channel map, configure channel endpoints/range on TX so that all channels go from ~1000 to ~2000. Set midpoint (default 1500), trim channels to 1500, configure stick deadband, verify behaviour when TX is off or out of range.
IMPORTANT: Before flying read failsafe chapter of documentation and configure failsafe.

Roll [A]	1500
Pitch [E]	1500
Yaw [R]	1500
Throttle [T]	885
AUX 1	1275
AUX 2	1500
AUX 3	1500
AUX 4	1500

Channel Map	RSSI Channel
AETR1234	Disabled

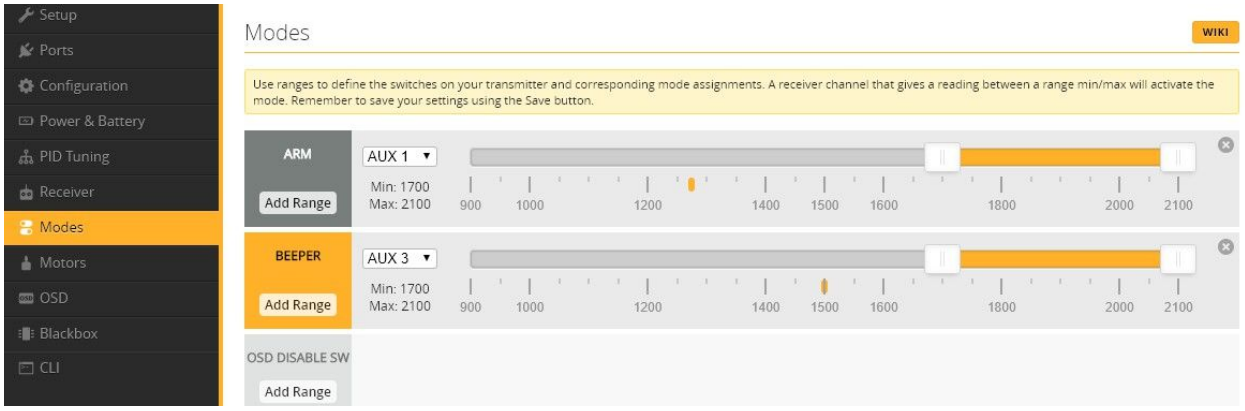
'Stick Low' Threshold	Stick Center	'Stick High' Threshold
1050	1500	1900

RC Deadband | Yaw Deadband | 3D Throttle Deadband

Figure 34. Channel Mapping and Configuration

The last configuration needed for this subsystem design is the ARM mode configuration. ARM mode is set to prevent users from sending any control signal to the quadcopter without recognitions. In the Modes page of the configurator, first the ARM mode has to be added. After that, the channel for the ARM mode and the range for triggering ARM mode can be set. In this design, the channel for ARM mode is set as AUX1, which is the 5th channel of the receiver. The range for ARM mode is from 1800 to 2000.

With above setting, in order to user the communication base to fly the quadcopter, the first step is to set Throttle channel value to below 1000, Roll, Pitch and Yaw channel value to 1500, and AUX 1 channel value to any value outside of the ARM mode range, such as 1100. The second stop to do is to set the AUX1 channel value so that it is in the ARM mode range, such as 1900. In this way, the ARM mode is triggered and the quadcopter is ready for further movement commands from the communication base.



Modes

Use ranges to define the switches on your transmitter and corresponding mode assignments. A receiver channel that gives a reading between a range min/max will activate the mode. Remember to save your settings using the Save button.

Mode	Channel	Range
ARM	AUX 1	Min: 1700 Max: 2100
BEEPER	AUX 3	Min: 1700 Max: 2100
OSD DISABLE SW		

Figure 35. ARM Mode Configuration

7. Performance Estimates and Results

At the end of this project, we had not achieved the goal of controlling the quadcopter by hand gestures, but we were very close to it. The communication base was able to interpret the gestures performed on the gesture controller and instruct the quadcopter how it should fly. The sections below are the detailed results for each subsystem.

7.1 Hand Controller

As the design specifications stated the hand controller must weigh less than 250 grams, be able to fall from 1.5 meters, have a flight time of at least 10 minutes, a setup time of fewer than 5 minutes, and be able to send positional hand data and special commands. The hand controller setup time is definitely within 5 minutes. The only setup that has to be done for the system to run properly is to flip the on-off switch on the LilyPad microcontroller. The fly time for the hand controller is determined by how long it can send data to the communication base. The hand controller from our tests can easily send positional information to the communication base for over an hour. The overall weight of the system is around 18 grams which are well below the 250-gram limit. Since the system is so light and the fabric is flat which allows the hand controller to not pick up speed when falling. It has been dropped from at least 1.5 meters and did not break. As outlined in the software design the implementation of the system has the ability to send positional data and specialized commands.

The system has the specification of having a less than 0.6-second display. This is so the pilot does not see a latency when sending an instruction to the quadcopter. One of the issues with the hand controller is the fact, from what we can measure, there is a 2.2-second latency. This is a much greater latency than we intended on having. If our system was operational there would be a major delay in the system from when the pilot would make the motion and the quadcopter would react. If this project would be continued this would have to be addressed with a change of the communication modules. Or figuring out a way to make the 433 MHz radio transmitter work more reliably by the use of a different library or another method.

7.2 Communication Base

The communication base has meet the minimal functionality we originally planned as it can transmit and receive data with rather short latency and good accuracy. We achieved a communication range of 2 meter between hand controller and base and 30 meter near line of sight in door communication between the video transmitter mounted on drone and base. Although the video signal has trouble penetrate through thick obstacles like doors, it can go through short wall in close distance. The video displayed on the monitor is clear overall and only degrade when a thick obstacle is presented.

However currently the communication is still operate on a breadboard, which is far from the final goal to fit the base into a small box. The biggest challenge currently is there are a lot of different components that require different voltages. If we can have a hardware design with better voltage regulation, packing the base would be a lot more easier. But overall, the requirement for the base is mostly actualized.

7.3 Quadcopter

Overall, the quadcopter subsystem has achieved the most important goals.

First, the quadcopter is able to fly just like a traditional quadcopter product, and it supports the four basic movements: roll, yaw, pitch and throttle tested by a Frsky transmitter as well as the communication base. Also, quadcopter supports special commands, such as take off, hover and landing. This functionality is tested by the communication base with running arduino testing code. In terms of receiver accuracy, each receiver channel only has a error rate of 0.05%, which means according to the tests conducted so far, compared to the expected value for each channel, the actual received value for each channel only has upto 1 value difference out of the 2000 value range.

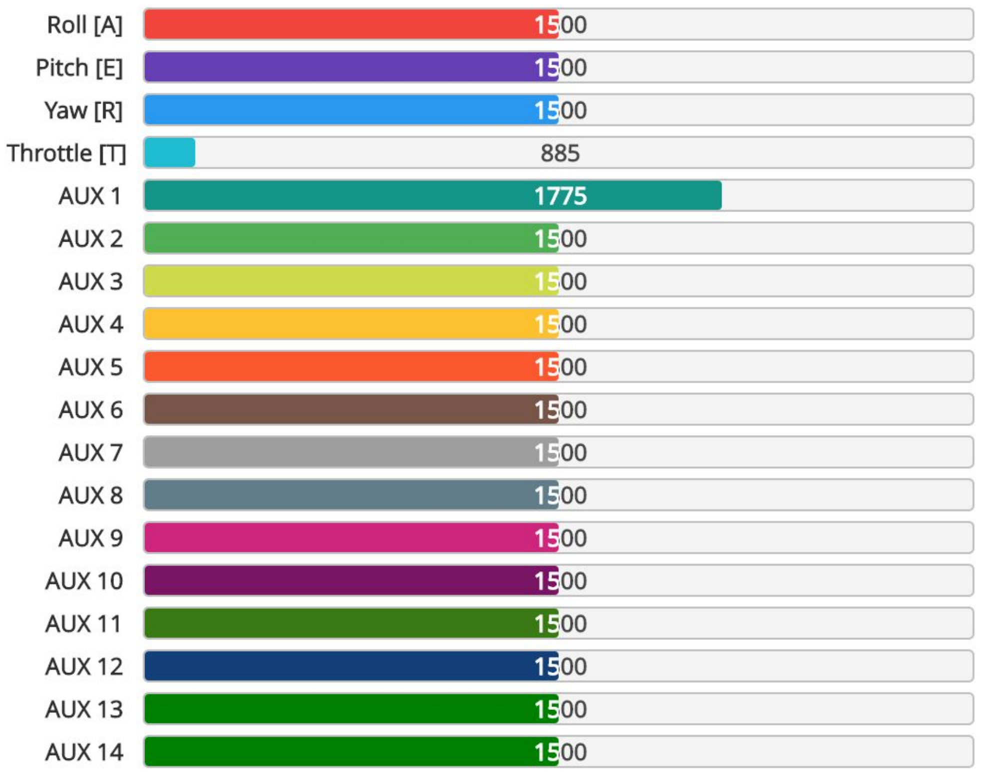


Figure 36. Receiver Accuracy Result

Second, the quadcopter subsystem is able to deliver the live video data back to the communication base, and the video data is able to be displayed on the monitor of the communication base.

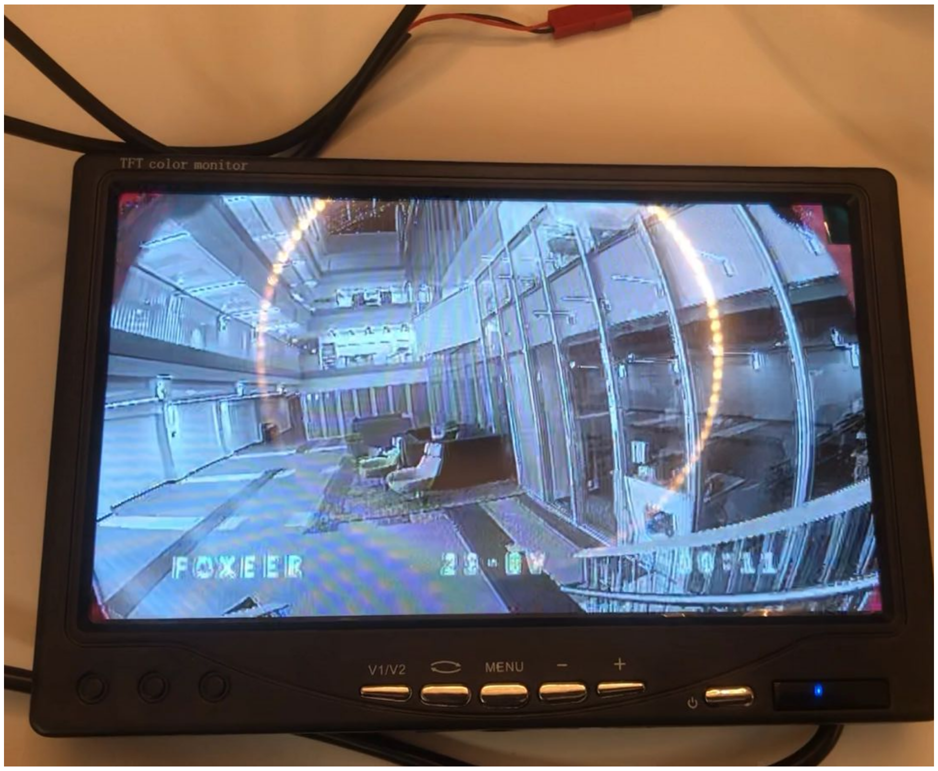


Figure 37. Video Live Streaming Test Result

Moreover, the size of the quadcopter subsystem is a 25.6x25.6x12.8 cm^3 ,which is much smaller than a 25x25x25 cm^3 space. The weight of the quadcopter is 570.19 gram, which is much lighter than the proposed 1.5 kg. In addition, it only takes around 45 minutes to fully charge the quadcopter subsystem, which is much shorter than the proposed 3 houses charging time. In terms of setting up time, it only takes less than 30 seconds to pair up the quadcopter and the communication base, which is much shorter than the proposed 5 minutes.

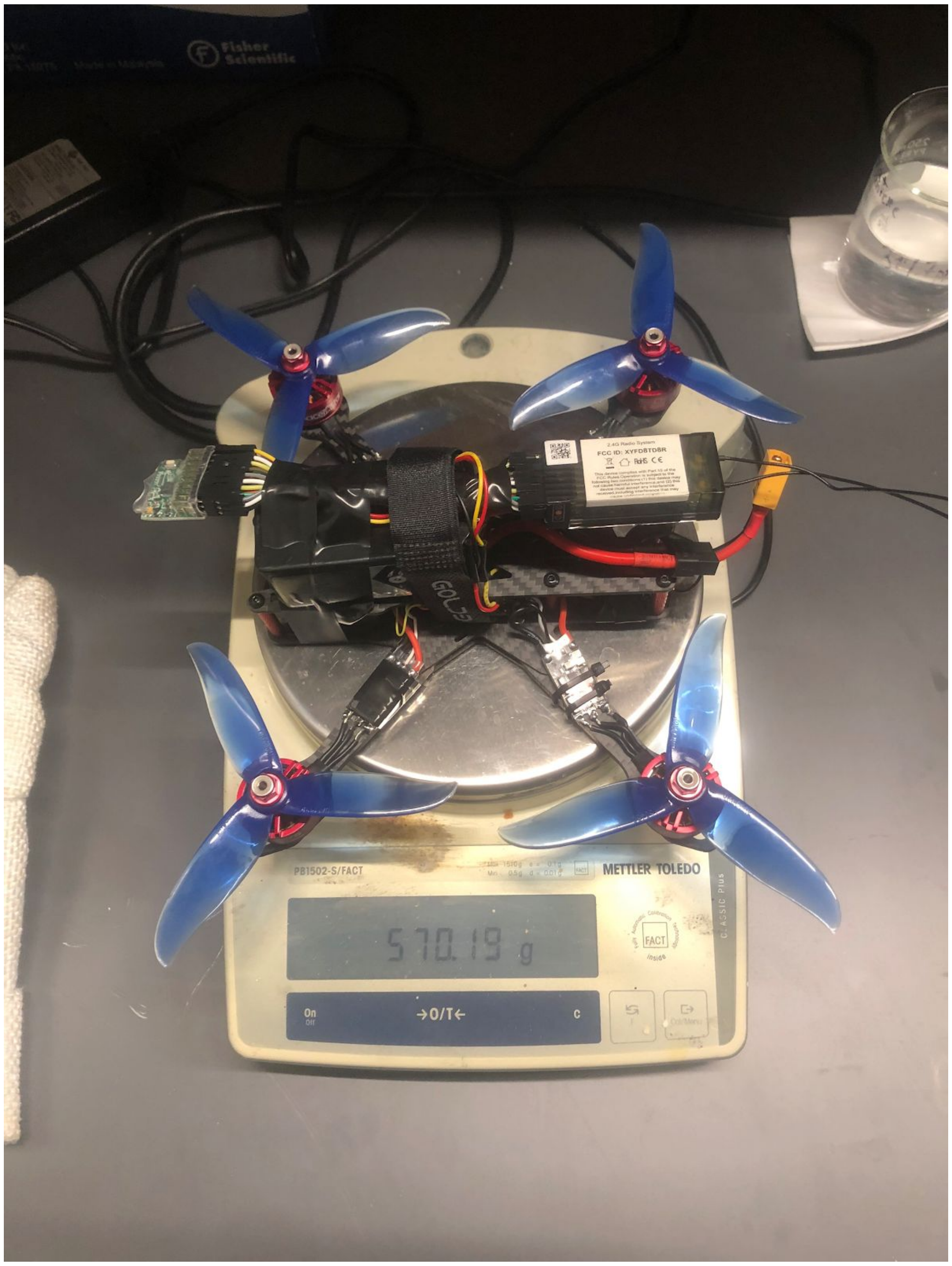


Figure 38. Weight Result for the Quadcopter

However, there is one design goal not achieved for current design and implementation. Even though communication between the communication base and the quadcopter subsystem is almost immediate, the communication delay between the gesture controller and the quadcopter is around 2.2 seconds, which is much longer than the proposed 0.6 second whole system delay.

Due to the broken components and long waiting period for receiving new components at in end of this project, the maximum speed and the total operation time for this subsystem are not tested yet.

8. Production Schedule

8.1 Overview

The overall production schedule for the gesture-controlled quadcopter system was to test the individual components of each subsystem, then if necessary create the software for controlling the subsystem and then finally integrate the three subsystems together. Below shows the pert diagram for the whole system.

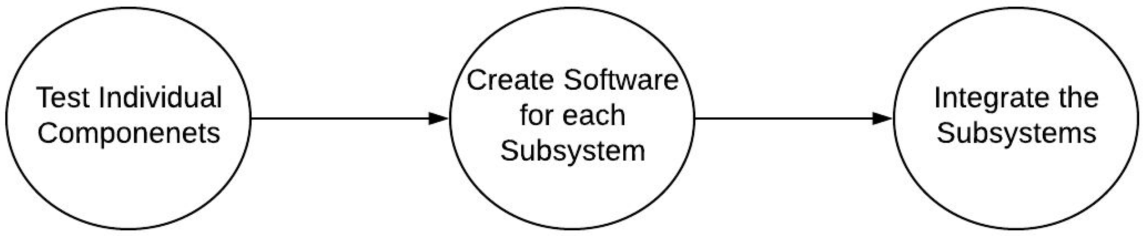

Test Individual Componenets
Create Software for each Subsystem
Integrate the Subsystems

Figure 39. Pert Diagram for the Overall Production Schedule

8.2 Hand Controller

The phases of the hand controller production schedule were hardware testing and implementation, software production, and full system integration.

8.2.1 Hardware

The production schedule for the hardware part of the hand controller we wanted to test each component individually. This was done by using prototyping techniques, such as alligator clips and single core wires instead of the sewn circuit board. This allowed for the testing of the buttons, transmitter, flex sensor, and 9-DOF board before creating a circuit that would have to be resewn if a connection needed to be changed. The buttons worked well with the alligator clips and were able to control the led which meant that they could trigger an event. The flex sensor was a bit more complex because it was a voltage divider and setting it up as a prototype was difficult. The same issues arose when testing the 9-DOF board the pins were close together and the alligator clips would touch and ground the data pins. The next step is creating the actual sewn circuit using the components mention before. The buttons had to be set up with an internal pull-up resistor because when the resistor was sewn into the system it did not read the correct state of the button. The flex sensor, once the correct second resistor was calculated, worked with the sewn system was affected by the fact that skin was a conductor and worked seamlessly when the wires were insulated. The 9-DOF board had it's difficulty when in prototyping but when sewn into the circuit it would not take any readings. Though many different sewing configurations were attempted the system would not work with the conductive thread. The component needed to have wires soldered into the pin so the data could be read correctly. The last component is the radio transmitter, this component worked when in the prototype stage and even more consistently when the system was a sewn circuit.

8.1.2 Software

After the components had been tested and were working with the LilyPad USB Plus the software that was used to test the components had to be integrated and modified to work together. The button and flex sensor was simple to integrate after learning the LilyPad did not have an interrupt pin beside the SDA and SCL. They both used software based interrupts which are discussed in the software design section of the hand controller. The 9-DOF freedom board used the LSM9DS1 Library, which as described in the software design has functions that can be used to read the data in. When integrating it into the system it just needed to be placed in the correct loop such that it only read data when the pilot wanted it to. The last component considered was the communication it took us a long time to decide on what was going to be sent and how it was going to be sent. The RadioHead library can only take information as multiples of 8 bits, meaning you can send a list with four 8 bit integers as an example. We were originally going to send the information as strings but we wanted to cut down on the information sent to cut down on the latency of the system. That's when we decided to send 8-bit unsigned integers, we spoke of how the data is formatted in the software design. The information was integrated into the system and the function to send information was set up to send positional data as frequently as it could if the system was enabled and to send the special commands when triggered. The software was complete and worked with every component of the system as intended.

8.2.3 Full System

The last part of the system production schedule was to integrate the three subsystems together. This was done with the hand controller by testing the communication to the base and making sure that the values computed were the values sent. As the communication base has described the values were then used and converted into signals that could be sent to the quadcopter as commands.

8.3 Communication Base

The work required by the communication base is a bit fragmented as the base has working pieces with both two other parts. The different components and task also furthered the fragmentation. The scheduling is rather simple: layout the infrastructure of a functional communication from hand to base to quadcopter first, help the other two parts when needed, and develop the user intention interpretation algorithm at the end.

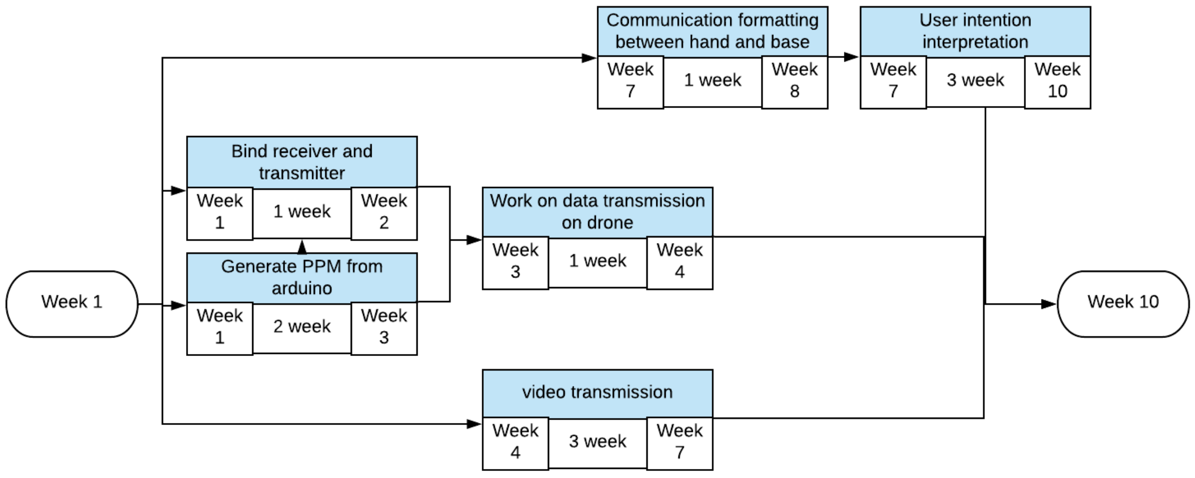

Communication formatting between hand and base
Week 7
1 week
Week 8
User intention interpretation
Week 7
3 week
Week 10
Bind receiver and transmitter
Week 1
1 week
Week 2
Work on data transmission on drone
Week 3
1 week
Week 4
Generate PPM from arduino
Week 1
2 week
Week 3
Week 1
Week 10
video transmission
Week 4
3 week
Week 7

Figure 40. Pert chart for communication base

8.4 Quadcopter

Before week 6, the work progress of the quadcopter subsystem were mainly towards two goals: control of motors from the flight controller, and receiving correct control signals form the communication base. On week 6, the two control paths were assembled into one, and the main focus shifted to how to control the quadcopter from the communication base. On week 9, due to the unexpected accident with the radio receiver of the quadcopter, the control of the quadcopter from the gesture controller could not be implemented, and the focus of the work shifted to tests for individual implemented features and functionalities of this subsystem. The details of the progress of each week is shown in the picture below.

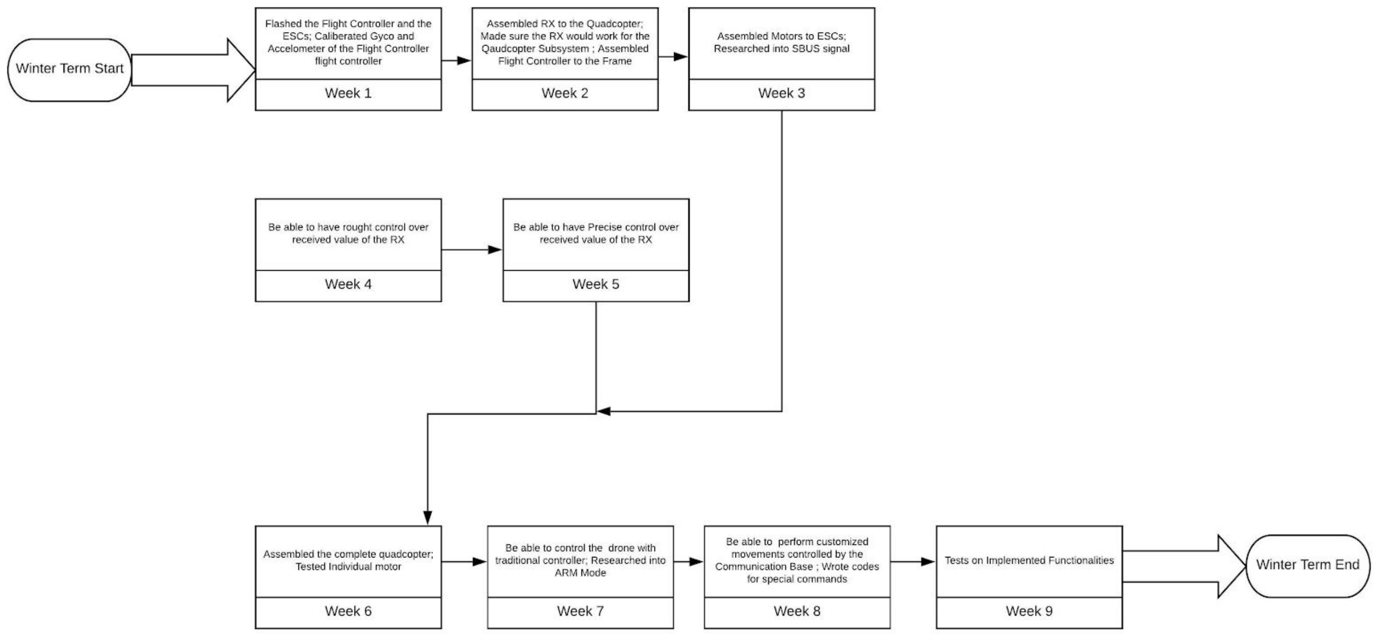

Winter Term Start
Flashed the Flight Controller and the ESCs; Caliberated Gyco and Accelometer of the Flight Controller flight controller
Week 1
Assembled RX to the Quadcopter; Made sure the RX would work for the Qaudcopter Subsystem ; Assembled Flight Controller to the Frame
Week 2
Assembled Motors to ESCs; Researched into SBUS signal
Week 3
Be able to have rought control over received value of the RX
Week 4
Be able to have Precise control over received value of the RX
Week 5
Assembled the complete quadcopter; Tested Individual motor
Week 6
Be able to control the drone with traditional controller; Researched into ARM Mode
Week 7
Be able to perform customized movements controlled by the Communication Base ; Wrote codes for special commands
Week 8
Tests on Implemented Functionalities
Week 9
Winter Term End

Figure 41. Precedence Diagram for the Quadcopter Subsystem

9. Cost Analysis

The total cost of the quadcopter was around $730, $437 was covered by the SRG funding we received and the rest was covered by department funds. A full list of the components and the amount of each one bought can be seen below in the table. The amount is greater than anticipated this was because a few more expensive components were accidentally burnt out or faulty and needed replacement. Quadcopters can run from $10 to over $1,000 so a professional quadcopter being in the $400 to $500 range isn't unreasonable. Though for application for police force using these systems it might be more expensive than expected.

Part of the System	Name of Component	Vendor	Number Ordered	Cost Including Shipping	Total Spent on Part
Hand Controller	Lilypad USB Plus	Sparkfun	1	$28.45	$28.45
Hand Controller	Adafruit LSM9DS1 9-DOF Board	Adafruit	1	$22.70	$22.70
Hand Controller	Flex Sensor	Sparkfun	1	$10.44	$10.44
Hand Controller	Sewable Lilypad Button x 2	Amazon	1	$6.39	$6.39
Hand Controller	Conductive Thread x 3	Sparkfun	1	$11.34	$11.34
Hand Controller	Coin Cell Battery Holder	Amazon	1	$6.89	$6.89
Hand Controller	Coin Cell Batteries x 5	Amazon	1	$4.95	$4.95
Hand Controller Base	433Mhz RF Transmitter and Receiver Module	Amazon	1	$10.55	$10.55
Quadcopter	Foxeer Falkor FPV Camera	getFPV	1	$39.90	$39.90
Quadcopter	FPV Transmitter	Amazon	3	$19.99	$59.97
Quadcopter	Frame: Martian II	Amazon	1	$26.99	$26.99
Quadcopter	Flight Controller: Kakute F4 V2 AIO FC	Google Express	1	$40.99	$40.99
Quadcopter	Motors: Racerstar x 4	Banggood	1	$42	$42
Quadcopter	Racerstar 30A V2 x 4	banggood	1	$50.35	$50.35
Quadcopter	Propeller: DALPROP T5045C Cyclone	banggood	3	$3	$9
Quadcopter	Battery: XILO 1500mAh 4s 100c Lipo Batter	getFPV	1	$22.99	$22.99
Quadcopter	Battery Charger	getFPV	1	$12.99	$12.99
Quadcopter	Signal Converter Module SBUS-PPM-PWM (S2PW)	hobbyking	2	$10.42	$20.84
Base	Elegoo EL-CB-001 UNO R3 Board ATmega328P ATMEGA16U2	Amazon	1	$10.86	$10.86
Base	FrSky DHT 8ch DIY Telemetry Compatible Transmitter Module	hobbyking	2	$23	$46
Base	RC FPV Monitor 7 Inch 1024x600 LCD Display Video Screen HD Monitor	Amazon	1	$49.99	$49.99

Quadcopter	FrSky D8R-II PLUS 2.4GHz 8CH Receiver with Telemetery	hobbyking	2	$27.20	$54.40
Quadcopter	pwm to sbus translator	ebay	1	$20.75	$20.75
Base	5.8G 32CH FPV Wireless Av Receiver W/ Led Channel Display Rc832 for RX	Amazon	1	$21.45	$21.45
Base	GOLDBAT 4S 1500mAh 100C 14.8V Lipo Battery Pack with XT60 Plug for RC Car Boat Truck Heli Airplane UAV Drone FPV Racing	Amazon	1	$19.99	$19.99
Quadcopter	Padarsey 2 Pair XT60 Connector Female/Male W/Housing 10CM Silicon Wire 14AWG	Amazon	1	$6.99	$6.99
Base	FrSky D8R-II PLUS	Amazon	1	$36.50	$36.50
Base	Frsky DHT DIY transmitter module	Amazon	1	$36.08	$36.08
Total Spent					$730.84
SRG Funding					**$437.00**
Department Funding					$293.84

Table 9. Cost List for the System

10. User's Manual

Before using the quadcopter, connect the flight controller to a computer with Betaflight Configurator and calibrate the accelerometer on the quadcopter.

If it is the first time the user uses the system, they need to bind the transmitter on the communication base to the receiver on the quadcopter. The user need to follow the given instructions:

1. Power off other the drone and the communication base
2. Turn on the communication base while holding the transmitter's binding button, the led light of the transmitter will start flashing and beeping.
3. Turn on the quadcopter while holding the binding button of the receiver, both the red and green led light on the receiver will turn on to indicate a successful binding.
4. Turn off both devices. Turn on the two devices. Now the red light on the receiver will emit a constant red light to show a constant data flow

Once the binding is done, the transmitter and the receiver are paired up until the next binding process, which follows the same instruction as above.

To turn on the hand controller you simply use the switch on the LilyPad USB Plus. There is no pairing necessary between the communication base and the hand controller. Below is a table of the instructions and the figure for gesture controls.

Instruction	Action
Take Off	System is landed bend finger with flex sensor greater than 90 degrees
Stop	Press the button closest to the snap on the palm on your hand
Hover	Press the button furthest from the snap
Increase Throttle	While enabled (light is green) bend the flex sensor greater than 90 degrees
Decrease Throttle	While enabled (light is green) bend the flex sensor greater than 90 degrees and flip the hand so the LilyPad is parallel to the floor

Table 10. Instructions for using the Special Commands

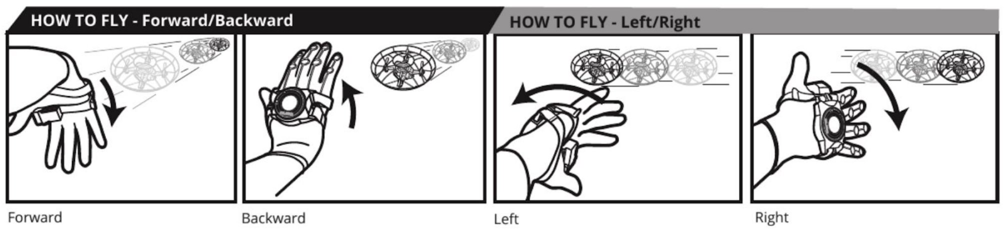


Figure 42. Hand Gestures Instruction for Aura

11. Discussion & Conclusions

Current UAV technology allows quadcopters to become helpful tools for many scenarios. For instance, a quadcopter with a camera could be used by the police to survey the inside of a building to avoid a dangerous situation. However, features of the joystick-button controller such as requiring long training time and demanding both hands for operations are preventing quadcopters to be applied to those scenarios. Thus we designed and implemented a gesture-controlled quadcopter to solve this problem.

The gesture-controlled quadcopter system is separated into three subsystems: the hand controller, communication base and the quadcopter. The hand controller is a glove-like controller that sends the positioning information to the communication base. The communication base is designed to figure out the current gesture based on the information from the hand controller, and send corresponding signals to the quadcopter . The quadcopter is designed to fly according to the instructions of the communication base and send back live video content.

As results for the overall system, the communication base was able to successfully interpret most of desired gestures from the hand controller. It could also control the quadcopter to perform basic movements as well as special commands such as emergency stop. However, because the receiver on the quadcopter broke on week 8, the project implementation stopped here, and we were not able to test the functionality of controlling the quadcopter by the hand controller.

As results for the hand controller, it was able to communicate with the base and send positional data of the hand along with special commands we designed. It is lightweight, has a setup time under 5 minutes, and has an operational time of over an hour before it needs to be charged. Though it does not meet the system's goal of having a latency under 2.2 seconds recommendations for fixing that can be seen in the next subsection. For the communication base . For the quadcopter, it met the most important goals: be able to conduct basic quadcopter movements, and be able to conduct special command movements. However, there are some goals we had not tested yet such operation time and maximum speed due to the broken radio receiver.

At the end of this project, we concluded that most of the latency exists in this system comes from the communication between the hand controller and the communication base because of the low-frequency radio communication between them. Although the responsiveness of the flight controller was a important standard for choosing it, the delay in the quadcopter itself becomes ignorable. Compared to quadcopter which could only do the basic four movements of quadcopters in Yucan Liu's single-handed gesture controller project[9], the quadcopter in our project is more flexible since it supports customized combination of movements. Those customized movements would contribute hugly to ease of flying a quadcopter. Indeed, we believe that the flexibility comes from the open-source flight controller.

11.1 Recommendations

For the overall system, when it is fully functioning in the future, people who are working on it should ask target customers to operate this system and give feedbacks about its different aspects such as ease of usage and responsiveness of the whole system. In this way, this project would have a bigger chance to become an actual helpful product in the future. There are also recommendations for each subsystems, and they are discussed in the following subsections.

11.1.1 Hand Controller

One of the greatest problems that the hand controller causes are latency. This is a problem because if you make a gesture or signal a special command you want the system to react as close to instantaneously as possible. We researched that a 0.6-second latency would be short enough that a human would not recognize there being any latency. The 2.2-second latency, we believe, comes from the radio communication we use. A recommendation would be to switch to the XBee communication system. This was originally something we considered as an option but we believed the system would not be compatible with the LilyPad system and add an unnecessary complication to the system. The XBee modules are designed to be low latency communication which would solve the problem with the current component we have. As for being incompatible with our system, we found LilyPad created an adaptor module specifically designed for XBee components.

The other part of the subsystem that can contribute to the latency that is produced in the system is the fact that the interrupts had to be software based interrupts. This was because even though the LilyPad USB uses an ATMega32 chip it uses a limited number of the pins it could. When designing the system a misreading of the schematic lead us to believe that the system would have the ability to set up hardware-based interrupts. Upon further examination, the only pins that allow for hardware-based interrupts are the SDA and SCL pins which are needed for the 9-DOF board. If this was realized earlier we would have used the LilyPad Arduino 328 Main

Board which has the full pinout of the ATmega328 chip. This full pinout would allow for the software based interrupts to be changed to hardware-based interrupts that would cut down the latency of the system.

11.1.2 Communication Base

There are a few problems that the base needs to improve. First is the delay and occasional data missing for the data transmission between hand controller and communication base as mentioned in the previous section. Although the exact impact of the delay on the system in general is still unknown, it is a drawback we want to avoid as much as possible.

The second problem is the noise in the gyroscope. We think the wrong reading is due to the poor calibration of the device instead of its accuracy. Thus an effective way to calibrate the gyroscope is needed. Although our current algorithm still function normally without gyroscope, the addition of it can definitely open a lot more option to the gesture.

And this leads us to the last problem, gesture design. Sadly currently our gesture setting only support three out of four basic movements. The gesture to change yaw is not implemented at all. In the same time, we can not change different kind of movement in the same time. We can change roll and pitch together, but the control for throttle is seperate. In the same time, we need to be aware that designing the gestures are harder than it seems at first. It need to map four degree of freedom to the gesture on a single hand, while considering about the intuitiveness for user, the possible gesture that be detected by the sensor and not too complicated for the base to process to cause delay.

11.1.3 Quadcopter

Although the quadcopter met the major goals of this system, it could be improved in a couple aspects. First in order to set the maximum speed for the quadcopter subsystem, the ESCs can be configured to limit the top speed of the quadcopter using a racerstar ESC configurator. Also, the betaflight flight controller is an open-sourced processor, which grants users a large freedom of customization of new features. In the future, researches on this subsystem should research deeper into what other features they can develop for it with this process. Last, in order to develop more advanced special command such as auto landing, more sensors could be installed to this subsystem.

11.2 Lessons Learned

Since this project was a group project it included the most difficult aspect of projects, people. Working a group of three leads to difficulty with communication, work separation and differentiating opinions. One of the greatest lessons learned was the ability to communicate and compromise on aspects of the design and implementation. Having three minds allows for more brain power solving the problem at hand but it can also lead to three different ways of approaching the problem that may not be compatible. This project has made us sit down and explain our processes to see where we could collaborate our ideas.

Another lesson that this team had to learn was the importance of buying extra components. When creating large projects such as this with components we have never worked with before it is important to prepare for the inevitable breaking of components. This means buying extras of components that are difficult to find or sourcing from a shop that ships products quickly. Since we had several components that were bought from foreign suppliers getting replacements would halt the building process for weeks.

12. References

[1]“Adafruit LSM9DS1 Accelerometer Gyro Magnetometer 9-DOF Breakout.” *Memory Architectures | Memories of an Arduino | Adafruit Learning System*, Adafruit, learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout /arduino-code.

[2] “AURA UNBOXED.” *Aura*, Cide Interactive SLU , 2017, aura-drone.com/us/.

[3] “Bluefruit LE - Bluetooth Low Energy (BLE 4.0) - nRF8001 Breakout.” *Adafruit Industries Blog RSS*, 17 July 2014, www.adafruit.com/product/1697?gclid=EAIaIQobChMIqbmpwPzg3gIVySSGCh2BpAW2EAQYASABEgJnNPD_BwE.

[4]“Choosing a LilyPad Arduino for Your Project.” *Sparkfun*, SparkFun Electronics, learn.sparkfun.com/tutorials/choosing-a-lilypad-arduino-for-your-project#resources-and-going-further.

[5]“Drone Services: Industrial - Commercial - Public Safety.” *In Sky Aerial LLC*, inskyaerialservices.com/.

[6] “FACTS & FIGURES Causes of Law Enforcement Deaths.” *Bertillon System of Criminal Identification*, National Law Enforcement Officers Memorial Fund, 15 Mar. 2018, www.nleomf.org/facts/officer-fatalities-data/causes.html.

[7]“Indoor Drones Open Up New Business Opportunities.” Unmanned Aerial, Zackin Publications Inc, 23 Apr. 2018, unmanned-aerial.com/indoor-drones-open-up-new-business-opportunities.

[8]LII Staff. “Fourth Amendment.” *LII / Legal Information Institute*, Legal Information Institute, 10 Oct. 2017, www.law.cornell.edu/constitution/fourth_amendment.

[9] Liu, Yucan. *SINGLE-HAND GESTURE CONTROLLER FOR QUADCOPTERS*. Bachelor's thesis, Union College, 2017. 2-65.

[10] Margaritoff, Marco. “*Drones in Law Enforcement: How, Where and When They're Used.*” *The Drive*, Time Inc, 13 Oct. 2017, www.thedrive.com/aerial/15092/drones-in-law-enforcement-how-where-and-when-theyre-used.

[11]"Naza-M V2 Quick Start Guide V 1.28 ." *Http://Dl.djicdn.com*, DJI, 6 Dec. 2015, dl.djicdn.com/downloads/nazam-v2/en/NAZA-M_Quick_Start_Guide_v1.28_en.pdf.

[12] "Pixhawk Mini." *Basic Concepts · PX4 User Guide*, PX4 Dev Team, 19 Nov. 2018, docs.px4.io/en/flight_controller/pixhawk_mini.html.

[13]Schreck, Ben, and Lee Gross. "Gesture Controlled UAV Proposal." *Web.mit.edu*, 29 Oct. 2014, web.mit.edu/6.111/www/f2014/projects/leegross_Project_Proposal.pdf.

[14]Technology, Lorex. "Camera Resolution and TV Lines (TVL)." *Lorex Site - CA*, Lorex Technology Inc., www.lorextechnology.com/self-serve/camera-resolution-and-tv-lines-tvl-/R-sc2900040.

[15] Wertz, Sally. "Drone Laws." *Drone and Quadcopter*, Drone and Quadcopter, 11 Aug. 2016, droneandquadcopter.com/drone-laws/.

[16]Zhang, Dong, et al. " Dual-Hand Gesture Controlled Quadcopter Robot." *IEE Xplore Digital Library*, 26 July 2017, ieeexplore.ieee.org/document/8028439/authors#authors.

[17] "#11031 Kakute F4 AIO (V2)." *Http://Www.holybro.com*, Holybro, www.holybro.com/manual/HolybroKakuteF4AIOV2Manual_v1.0.pdf.

[18]"2010 Census Urban and Rural Classification and Urban Area Criteria." *Census Bureau QuickFacts*, United State Census Bureau, 1 Sept. 2012, www.census.gov/geo/reference/ua/urban-rural-2010.html.

[19] Github - Betaflight, https://github.com/betaflight/betaflight/wiki, 5 Dec. 2019.

13. Appendices

Appendix 1 - Complete Parts List with Prices

Part of the System	**Name of Component**	**Vendor**	**Cost Including Shipping**
Hand Controller	Lilypad USB Plus	Sparkfun	$28.45
Hand Controller	Adafruit LSM9DS1 Accelerometer + Gyro + Magnetometer 9-DOF Breakout Board	Adafruit	$22.70
Hand Controller	Flex Sensor 2.2"	Sparkfun	$10.44
Hand Controller	Sewable Lilypad Button x 2	Amazon	$6.39
Hand Controller	Conductive Thread x 3	Sparkfun	$11.34
Hand Controller	Coin Cell Battery Holder	Amazon	$6.89
Hand Controller	Coin Cell Batteries x 5	Amazon	$4.95
Hand Controller & Communication Base	433Mhz RF Transmitter and Receiver Module	Amazon	$10.55
Quadcopter	Foxeer Falkor 1200TVL 1.8mm FPV Camera - Limited Edition White	getFPV	$39.90
Quadcopter	FPV Transmitter, EACHINE VTX03 Super Mini FPV Transmitter 5.8G 72CH 0/25mW/50mw/200mW Switchable Transmission	Amazon	$19.99
Quadcopter	Frame: Martian II	Amazon	$26.99
Quadcopter	Flight Controller: Kakute F4 V2 AIO FC	Google Express	$40.99
Quadcopter	Motors: Racerstar 2207 * 4	Bang Good	$42.28
Quadcopter	Racerstar 30A V2 * 4	banggood	$50.35
Quadcopter	Propeller: DALPROP T5045C Cyclone	banggood	$3
Quadcopter	Battery: XILO 1500mAh 4s 100c Lipo Batter	getFPV	$22.99
Quadcopter	Battery Charger: EV-Peak E3 Falcore Edition 35W 3A LiPo Battery Balance Charger	getFPV	$12.99

Quadcopter	Signal Converter Module SBUS-PPM-PWM (S2PW)	hobbyking	$10.42
Base	Elegoo EL-CB-001 UNO R3 Board ATmega328P ATMEGA16U2 with USB Cable for Arduino	Amazon	$10.86
Quadcopter	Padarsey 2 Pair XT60 Connector Female/Male W/Housing 10CM Silicon Wire 14AWG	Amazon	$6.99
Base	FrSky DHT 8ch DIY Telemetry Compatible Transmitter Module	hobbyking	$23
Base	RC FPV Monitor 7 Inch 1024x600 LCD Display Video Screen HD Monitor NO Blue Screen with Hood Sun Shield and XT60 to Deans Connector	Amazon	$49.99
Base	FrSky D8R-II PLUS 2.4GHz 8CH Receiver with Telemetry	hobbyking	$27.20

Appendix 2 - Full Hand Control Code

```
#include <RH_ASK.h>
#ifdef RH_HAVE_HARDWARE_SPI
#include <SPI.h>
#endif
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_LSM9DS1.h>
#include <Adafruit_Sensor.h>
//9 DOF Sensor
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1();

#define LSM9DS1_SCK 10
#define LSM9DS1_MISO 12
#define LSM9DS1_MOSI 11
#define LSM9DS1_XGCS 6
#define LSM9DS1_MCS 5

void setupSensor()
{
  // 1.) Set the accelerometer range
  lsm.setupAccel(lsm.LSM9DS1_ACCELRANGE_2G);
  // 2.) Set the magnetometer sensitivity
  lsm.setupMag(lsm.LSM9DS1_MAGGAIN_4GAUSS);
  // 3.) Setup the gyroscope
  lsm.setupGyro(lsm.LSM9DS1_GYROSCALE_245DPS);
}
    int SpecialMessage = 50;
    float cgx = 0.0;
    float cgy = 0.0;
    float cgz =0.0;
    float cax = 0.0;
    float cay =0.0;
    float caz = 0.0;
    uint16_t cgxi = 0;
    uint16_t  cgyi = 0;
    uint16_t  cgzi = 0;
    uint16_t  caxi = 0;
    uint16_t  cayi = 0;
    uint16_t  cazi = 0;

//Transmitter
RH_ASK driver(2000, 8, 6, 0); //Recieve, Transmit, Push to talk
uint8_t msg[10] = {SpecialMessage,caxi,cayi,cazi,(cgxi>>8),cgxi,(cgyi>>8),cgyi,(cgzi>>8),cgzi};

//FlexSensor
const int FLEX_PIN = A9; // Pin connected to voltage divider output
const float VCC = 3.25; // Measured voltage of Ardunio 3.3V line
const float R_DIV = 35700.0; // Measured resistance of 36k resistor
const float STRAIGHT_RESISTANCE = 23683.90; // resistance when straight
const float BEND_RESISTANCE = 46185.88; // resistance at 90 deg

//LED
int RGB_red = 12;
int RGB_green = 13;
int RGB_blue = 14;
int Stop = 7;
int Hover = 8;
int enable = 0;

int HoverState =0;
int StopState =0;

void setup() {
  //Transmitter
  driver.init();

  Serial.begin(115200);
  //9 DOF Sensor
  lsm.begin();
  setupSensor();

  //Buttons
  pinMode(Stop, INPUT_PULLUP);
  pinMode(Hover, INPUT_PULLUP);
  //FlexSensor
  pinMode(FLEX_PIN, INPUT);
  //LED
  pinMode(RGB_red, OUTPUT);
  pinMode(RGB_green, OUTPUT);
  pinMode(RGB_blue, OUTPUT);
}

  void hoverInterrupt(){
  if(HoverState == 0 && StopState ==0&&enable ==1){
     msg[0] = 85;
     digitalWrite(RGB_green, LOW);
     enable = 0;
     HoverState = 1;
     for(int i=0; i<3; i++){
      driver.send(msg, 10);
      driver.waitPacketSent();
      delay(200);
     }
  }
  else if(HoverState == 1 && StopState ==0){
    digitalWrite(RGB_green, HIGH);
    msg[0] = 0;
    enable = 1;
    HoverState = 0;
  }

}

void stopInterrupt(){
     msg[0] = 170;
     digitalWrite(RGB_green, LOW);
     enable = 0;
     StopState = 1;
     HoverState = 0;

      for(int i=0; i<3; i++){
      driver.send(msg, 10);
      driver.waitPacketSent();
      delay(200);
     }
     msg[0] = 50;

}

void loop() {
  if (digitalRead(Stop)== LOW){
      stopInterrupt();
    }
    if (digitalRead(Hover)== LOW){
      hoverInterrupt();
    }

  if(enable == 0 && HoverState ==0){
    int flexADC = analogRead(FLEX_PIN);
    float flexV = flexADC * VCC / 1023.0;
    float flexR = R_DIV * (VCC / flexV - 1.0);
     if(flexR> BEND_RESISTANCE){
       msg[0] = 255;
       enable = 1;
       StopState = 0;
       digitalWrite(RGB_green, HIGH);
     }
  }
  if(enable ==1){
    digitalWrite(RGB_green, HIGH);
  }
  driver.send(msg, 10);
  driver.waitPacketSent();
  delay(200);

  while(enable ==1){
    if (digitalRead(Stop)== LOW){
```

```
    stopInterrupt();
    break;
  }
  if (digitalRead(Hover)== LOW){
    hoverInterrupt();
    break;
  }
  lsm.read();

  sensors_event_t a, m, g, temp;

  lsm.getEvent(&a, &m, &g, &temp);

  int SpecialMessage = 0;
  float cgx = g.gyro.x*(32768/286.8) + 32768;
  float cgy = g.gyro.y*(32768/286.8) + 32768;
  float cgz = g.gyro.z*(32768/286.8) + 32768;
   if (digitalRead(Stop)== LOW){
    stopInterrupt();
    break;
  }

  float cax = a.acceleration.x*(128/19.6) + 128;
  float cay = a.acceleration.y*(128/19.6) + 128;
  float caz = a.acceleration.z*(128/19.6) + 128;
   if (digitalRead(Stop)== LOW){
    stopInterrupt();
    break;
  }
  if (digitalRead(Hover)== LOW){
    hoverInterrupt();
    break;
  }

  uint16_t cgxi = (uint16_t ) cgx;
  uint16_t  cgyi = (uint16_t ) cgy;
  uint16_t  cgzi = (uint16_t ) cgz;

   if (digitalRead(Stop)== LOW){
    stopInterrupt();
    break;
  }
  if (digitalRead(Hover)== LOW){
```

```
  hoverInterrupt();
  break;
}
uint16_t  caxi = (uint16_t ) cax;
uint16_t  cayi = (uint16_t ) cay;
uint16_t  cazi = (uint16_t ) caz;
if (digitalRead(Stop)== LOW){
  stopInterrupt();
  break;

}
if (digitalRead(Hover)== LOW){
  hoverInterrupt();
  break;
}
msg[0] = 0;
msg[1] = caxi;
msg[2] = cayi;
msg[3] = cazi;
msg[4] = (cgxi>>8);
msg[5] = cgxi;
msg[6] = (cgyi>>8);
msg[7] = cgyi;
msg[8] = (cgzi>>8);
msg[9] = cgzi;
int flexADC = analogRead(FLEX_PIN);
float flexV = flexADC * VCC / 1023.0;
float flexR = R_DIV * (VCC / flexV - 1.0);
 if(flexR> BEND_RESISTANCE){
    msg[0] = 1;
 }
     if (digitalRead(Stop)== LOW){
  stopInterrupt();
  break;
}
if (digitalRead(Hover)== LOW){
  hoverInterrupt();
  break;
}
driver.send(msg, 10);
driver.waitPacketSent();
if (digitalRead(Stop)== LOW){
  stopInterrupt();
```

```
      break;
    }
    if (digitalRead(Hover)== LOW){
      hoverInterrupt();
      break;
    }

  }


}
```

Appendix 3 - Testing code for Take off, Hover and Landing with Emergency Stop

```
// PPM Code from https://code.google.com/archive/p/generate-ppm-signal/
// PPM Config
#define chanel_number 8  //set the number of chanels
#define default_servo_value 1500  //set the default servo value
#define PPM_FrLen 20000  //set the PPM frame length in microseconds (1ms = 1000µs)
#define PPM_PulseLen 300  //set the pulse length
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is negative
#define sigPin 13  //set PPM signal output pin on the arduino
// Serial Config
const byte numChars = 41;
char receivedChars[numChars];
char tempChars[numChars];
int receivedData[chanel_number];
const int buttonPin = 2;     // the number of the pushbutton pin
boolean newData = false;
boolean setUp = true;
boolean ifStop = false;
boolean EmergencyStop = false;
// Minimums
int motorMin = 1000;
/*this array holds the servo values for the ppm signal
 change these values in your code (usually servo values move between 1000 and 2000)*/
int ppm[chanel_number];
void setup(){
  // initialize the pushbutton pin as an input:
  pinMode(buttonPin, INPUT);// inputpin for emergency stop
  attachInterrupt(0, pin_ISR, CHANGE);
  Serial.begin(9600);
  //initialize servo middle pwm values
  for(int i=0; i<chanel_number-2; i++){
  ppm[i]= default_servo_value;
  }
  ppm[6] = motorMin;
  ppm[7] = motorMin;
  pinMode(sigPin, OUTPUT);
  digitalWrite(sigPin, !onState);  //set the PPM signal pin to the default state (off)
  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;
  OCR1A = 100;  // compare match register, change this
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  sei();
}
void loop(){
  Serial.print("Setup Value: ");
  Serial.println(setUp);
  Serial.println("ifStop Value: ");
  Serial.println(ifStop);
  if (setUp){
     ppm[0] = 1478;// analogToRange(A0, 0, 2000);
     ppm[1] = 1478;//analogToRange(A1, 0, 2000);
     ppm[2] = 900;//analogToRange(A2, 0, 2000);throttle value
     ppm[3] = 1478;//analogToRange(A3, 0, 2000);
     ppm[4] = 1000;//analogToRange(A4, 0, 2000);arm mode
     ppm[5] = 1478;//analogToRange(A5, 0, 2000);
     ppm[6] = 1478;//analogToRange(A6, 0, 2000);
     ppm[7] = 1478; //analogToRange(A7, 0, 2000);
     setUp = false;
     delay(3000);
     //ppm[4] = 1500;//analogToRange(A4, 0, 2000);arm mode
  }
  else{
    if (ifStop){
     ppm[4] = 1000;
     while(1){
     }
    }
    else{
     ppm[4] = 1900;
     Serial.println(ppm[4]);
     delay(500);
     for(int i=900;i<1100;i+=10){
       ppm[2] = i;
       delay(1000);
     }
     delay(15000);
     for(int i=1100;i>900;i-=10){
       ppm[2] = i;
       delay(1000);
     }
     ppm[4] = 1000;
     ifStop = true;
    }
  }
}

void pin_ISR() {

   EmergencyStop = true;

}

ISR(TIMER1_COMPA_vect){  //leave this alone
  static boolean state = true;

  TCNT1 = 0;

  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
     cur_chan_numb = 0;
     calc_rest = calc_rest + PPM_PulseLen;//
     OCR1A = (PPM_FrLen - calc_rest) * 2;
     calc_rest = 0;
    }
    else{
     if(EmergencyStop){
       if(cur_chan_numb == 4){
         OCR1A = (1000 - PPM_PulseLen) * 2;
         calc_rest = calc_rest + 1000;
         cur_chan_numb++;
       }
       else{
        OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
       calc_rest = calc_rest + ppm[cur_chan_numb];
       cur_chan_numb++;
       }
     }
     else{
     OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
     calc_rest = calc_rest + ppm[cur_chan_numb];
     cur_chan_numb++;
     }
    }
  }
}
// Serial functions below from http://forum.arduino.cc/index.php?topic=396450
void recvWithStartEndMarkers() {
    static boolean recvInProgress = false;
    static byte ndx = 0;
    char startMarker = '<';
    char endMarker = '>';
    char rc;
    while (Serial.available() > 0 && newData == false) {
       rc = Serial.read();
       if (recvInProgress == true) {
           if (rc != endMarker) {
               receivedChars[ndx] = rc;
               ndx++;
               if (ndx >= numChars) {
                   ndx = numChars - 1;
               }
           }
           else {
               receivedChars[ndx] = '\0'; // terminate the string
               recvInProgress = false;
               ndx = 0;
               newData = true;
           }
       }
       else if (rc == startMarker) {
           recvInProgress = true;
       }
    }
}

//============


void parseData() {      // split the data into its parts
    char * strtokIndx; // this is used by strtok() as an index
    strtokIndx = strtok(receivedChars,",");//tempChars,",");
    Serial.println(strtokIndx);
    receivedData[0] = atoi(strtokIndx);
    for (int i=1;i<chanel_number;i++) {
     strtokIndx = strtok(NULL, ",");
     receivedData[i] = atoi(strtokIndx);
    }
}
//============
void showParsedData() {
  for (int i=0;i<chanel_number;i++) {
    Serial.print(receivedData[i]);
    Serial.println();
  }
}

int analogToRange(int pinNum, int minInt, int maxInt){
  int val = analogRead(pinNum);
  double difference = double(maxInt - minInt);
  int result = int(difference*val/1024);
  return result;
}
```