**Autonomous Response Drone**

| A Project Report  Presented to  The Faculty of the Computer Engineering Department |
| --- |
| San Jose State University  In Partial Fulfillment  Of the Requirements for the Degree  Bachelor of Science in Computer Engineering |

| Copyright © 2017 |
| --- |
|  |
| ALL RIGHTS RESERVED |

| **APPROVED FOR THE COLLEGE OF ENGINEERING** |
| --- |
|  |
| Kaikai Liu, Project Advisor |
|  |
| Professor Rod Fatoohi, Instructor |
|  |
| Dr. Xiao Su, Computer Engineering Department Chair |

**Acknowledgements**

*Advisor*

Kaikai Liu

[Kaikai.liu@sjsu.edu](mailto:Kaikai.liu@sjsu.edu)

*Authors*

Ali Baqar

[alinbaqar@gmail.com](mailto:alinbaqar@gmail.com)

Anahit Sarao

[anahitsarao@gmail.com](mailto:anahitsarao@gmail.com)

[Github](https://github.com/s3nu) - <https://github.com/s3nu>

Maxwell Cheshier

[maxcheshier@gmail.com](mailto:maxcheshier@gmail.com)

Brian Josefowicz

[brian.josefowicz@sjsu.edu](mailto:brian.josefowicz@sjsu.edu)

Project Video Links:

[Drone Alpha and Beta Stage](https://www.youtube.com/watch?v=I5GarBK532Q)

<https://www.youtube.com/watch?v=I5GarBK532Q>

[Drone Assembly](https://www.youtube.com/watch?v=mrRnQqFHLaM)

<https://www.youtube.com/watch?v=mrRnQqFHLaM>

[Drone Testing and Flight](https://www.youtube.com/watch?v=yeomkZmfY4Q)

<https://www.youtube.com/watch?v=yeomkZmfY4Q>

**ABSTRACT**

**Autonomous Response Drone**

By Ali Baqar, Maxwell Cheshier, Brian Josefowicz, and Anahit Sarao

College campuses have always dealt with criminal offenses and are always attempting to make these campuses safer for students. Most of the crimes that take place are burglaries, sex offenses, and theft. The police’s only response to this situation is to send out officers to the scene of the crime in hopes of catching the assailant in the act. With police forces struggling to catch the assailants, this can lead to a general feeling of unsafety. A school which lacks safety could possibly see fewer students enrolling out of fear of the campus. Additionally, the only deterrents the police have for this type of behavior is the use of security cameras which can be ineffective for the job required of them. Many of the assailants are able to escape and remain uncaught while relying solely on the victim’s description of the assailant in hopes of finding them. The use of a device which can respond to the situation in time faster than the response speed of police on the ground as well as aid the individual being attacked would serve as both an aid to the police and a deterrent against the attacker.

The current problem faced by police forces in attending to the aid of individuals is their slow response speed. If they are lucky, there will already be a unit on campus, however, if not, a unit will have to be deployed from their station which would reduce response time and their effectiveness. This slow response time often results in the escape of the attacker where they can’t possibly identify him or her. The use of cameras can help to solve them but it’s possible that these cameras either aren’t sensitive enough, lack a proper field of view, or simply aren’t in that location. Additionally, the cameras can’t aid the victim while they are being attacked which does not help in hindering the assailant. When cameras aren’t present, there is no way to identify the attacker besides witness descriptions. These descriptions can be faulty or lacking in detail and ultimately don’t often result in apprehending the attacker.

The drone will act as a first responder to a potential crime happening on campus. The drone will be able to reach the destination of where an attack is taking place faster than the police can respond. When the button is pressed to signal the police to an emergency, the drone will be signalled to attend to the location specified. When the drone responds, it will capture images of the location where it was sent in the form of video or still images, in order to aid in capturing photographic evidence of the attacker. This will allow the drone to both deter attacks and aid in identifying and apprehending the aggressor through image capturing. The use of an autonomous drone will allow the police to mobilize without the need of a “pilot” as well as allowing the drone to fly according to its program avoiding possible human error. The drones are also cheaper and more expendable than police lives resulting in a safer and more economical form of police aid. This drone will serve as an invaluable police aid and also a criminal activity deterrent.

**Table of Contents**

**List of Figures**  **7**

**List of Tables 8**

**Chapter 1 Introduction 9**

* 1. Project Goals and Objectives 9
  2. Problem and Motivation 9
  3. Project Application and Impact 9
  4. Project Results and Deliverables 10

**Chapter 2 Background and Related Work 10**

* 1. Background and Technologies 10
  2. Literature Search 12
  3. State-of-the-art 13

**Chapter 3 Project Requirements 14**

3.1 Domain and Business Requirements 14

3.2 System (or Component) Functional Requirements 14

3.3 Non-functional Requirements 15

3.4 Context and Interface Requirements 15

3.5 Technology and Resource Requirements 16

**Chapter 4 System Design 17**

4.1 Architecture Design 17

4.2 Interface and Component Design 19

4.3 Structure and Logic Design 20

4.4 Design Constraints, Problems, Trade-offs, and Solutions 20

**Chapter 5 System Implementation 22**

5.1 Implementation Overview 22

5.2 Implementation of Developed Solutions 22

5.3 Implementation Problems, Challenges, and Lessons Learned 23

**Chapter 6 Tools and Standards 24**

6.1 Tools Used 24

6.2 Standards 25

**Chapter 7 Testing and Experiment 25**

7.1 Testing and Experiment Scope 25

7.2 Testing and Experiment Approach 26

7.3 Testing and Experiment Results and Analysis 28

**Chapter 8 Conclusion and Future Work 29**

8.1 Conclusion 29

8.2 Future Work 30

**References 32**

**Appendices 33**

**List of Figures**

Figure 1. Project Domain and Motivation 10

Figure 2. Mission Controller Layout 13

Figure 3. UML2 Activity Diagram 16

Figure 4. ARD Central Stack 20

Figure 5. Overall System Block Diagram 21

Figure 6. Diptrace Pinout of Full System 22

Figure 7. I2C Communication 35

Figure 8. UART Communication 35

Figure 9. Motor RPM Due to Applied Current 36

Figure 10. Ping Sensor Results 38

Figure 11. Flight Destinations for Tests 40

Figure 12. Bootup Tests 41

Figure 13. Hardware Mounting Chassis 43

Figure 14. Underside View of Mounted Hardware Chassis 43

Figure 15. Drone and Peripherals 43

Figure 16. Internal Drone Core 43

Figure 17. BitCraze BigQuad Expansion Board 43

Figure 18. uBlox GPS Module 43

Figure 19. Raspberry Pi Zero 43

Figure 20. Arduino Ping Sensor 43

**List of Tables**

Table 1. Parts List and Description 17

Table 2. Average Flight Times 30

Table 3. GPS Flight Information 33

**Chapter 1. Introduction**

* 1. **Project Goals and Objectives**

The autonomous drone had several goals in mind. The ultimate goal was to provide a safer college campus by deterring criminal activity as well as aiding in apprehending the criminals that commit these offences. To achieve this, the drone will operate autonomously when it receives a signal to fly to a location. Once the destination for the drone to attend to is chosen, the drone begins its flight takeoff procedures. After preflight checks of battery level and peripheral status are confirmed, the drone takes flight to the specified location. From that point, the drone records images of the attacker whether in still frames or video. This process will also occur simultaneously with flashing lights flashing which can deter an attacker. When the police have arrived, the drone the drone received a signal indicating for its return to base. The drone returned to base where it goes into a standby mode awaiting the next distress signal.

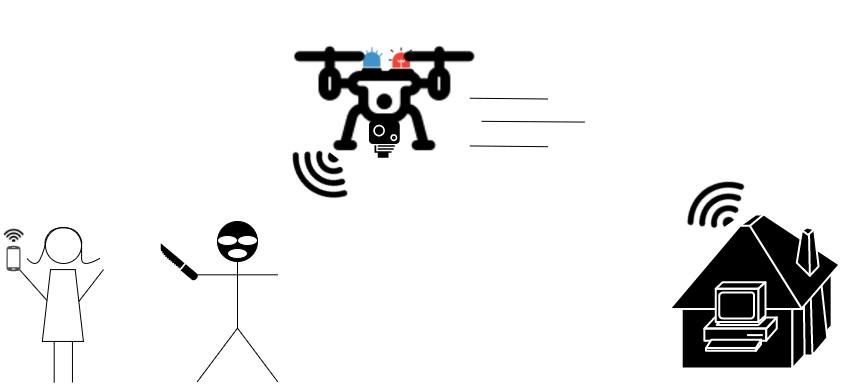
This drone has aimed to aid in the prevention of criminal activity on college campuses. Many of the activities that take place on college campuses can be prevented or prevented from escalating with the use of a fast response device aimed at alerting nearby individuals and alarming the attacker. With the increase in drone popularity and the advances in the field of automation, this project was a foreseeable step forward in the advances of security and automation.

* 1. **Problem and Motivation**

In the United States there has been a seemingly increase in terror attacks. More recently, there was a terror attack which resulted in a huge loss of life which occured in Las Vegas. This attack occurred at Mandalay Bay where the gunman was shooting from his hotel room. The gunman had placed surveillance cameras in his room in the case of police activity [15]. In the end, the police were not very useful in stopping the gunman as he was killed by his own gunshot. This attack is seen as something that could have been retaliated against more successfully if another apparatus besides brute force was used to determine the gunman’s location and preparations. The police could have employed a drone with a visual feed which would help them to gather real time intelligence on the shooter and his situation. This would allow the police to gain intelligence without sacrificing any officers and would also help the officers to operate quicker and more effectively.

Compared to a national scale, our campus can still be seen as a location where attacks happen regularly. San Jose State University is at the heart of downtown San Jose which is notorious for homelessness and criminal activity. Often this sort of activity can drift onto San Jose State’s campus and lead to crime that impacts students. While San Jose campus police have attempted to become more transparent and active with the criminal activity happening through SJSU alerts, this often leads to a greater feeling of unsafety and lack of police power. To deal with this, some changes must be made in how the police go about apprehending criminals and preventing any sort of criminal activity.

To properly act as a deterrent and aid for police, the drone caused the assailant to feel a sense of panic resulting in them fleeing the scene and additionally assisted the police in identifying the suspect. The drone caused this sense of alarm and panic through the use of lights and sounds that have the capability to alert bystanders and additionally assist the police in quickly acquiring the location of where the distress signal was sounded. The drone came to the aid of the victim and scared away the attacker and also provided crucial evidence to the police.



**Figure 1.** Domain of project is based on people in need. When a request is sent, coordinates are sent to the Drone to determine what location to fly to.

* 1. **Project Application and Impact**

The results of this project will develop a safer college campus where these drones are implemented. This will have a societal impact in creating a more enticing college experience where individuals don’t fear for their safety. With individuals not worrying about their safety, they will be able to focus properly on their academics resulting in more successful students and graduates. The use of these drones will also impact industry resulting in drones being utilized in more aspects of society. Drones may see use throughout cities in the delivering of goods or even see expanded use of their current application in police forces in urban areas. Currently, drones are seeing use by police forces in urban areas, however, these forces are manned which detracts from man power that could be used more effectively [1].

This drone can pave the way for additional drones for distress purposes. As it stands currently, the drones being used by military and police are used for riot control or bomb detection. While drones could currently be implemented to respond to distress signals, they would have to be operated manually. By being operated manually, you are spending significantly more resources on that drone operator compared to using a drone that works autonomously. An autonomous drone can avoid user error and will cost less long term.

The use of these drones may also have adverse effects on society. The use of drones may call into question the possible infringement on people’s fourth amendment rights. People may feel that they did not actively consent to the use of these drones for their surveillance purposes. This may result in possible legislation being passed either in favor or opposition to the use of drones and will have many implications on the future of drones.

* 1. **Project Results and Deliverables**

The drone has met several of our desired results and has failed to meet some that we had set out earlier in the project. The drone is able to provide video/photo capturing utilities through the use of a raspberry pi and attached camera. Additionally, the drone has the ability to take an input of a desired location and autonomously arrive at that specified location. When travelling to its destination, it utilizes sensors to detect objects and avoid those throughout flight. The drone controller has been integrated with the larger Xiro frame and can carry the payload of the added ESCs, lipo battery, and raspberry pi as well as other peripheral devices. This design is a minimum viable product and it is foreseeable that features can be improved upon to make this a large scale, usable device with real world applications.

There were some troubling problems that resulted in our changing of the MVP. We initially wanted to allow a user to send the “distress signal” from their phone and the drone would attend to that location even if the user was moving. This proved to be too hard in reality and strayed too much from the actual product. Implementing a phone application for different OSs is not something that can be done simultaneously with building the drone. Our next logical step was to introduce a beacon which could be placed throughout campus which would mimic the “distress signal” we had desired earlier but found that this would be unwieldy. The beacons could be destroyed and would be too much of a liability. It would also involve us making a beacon capable of communicating its location with the drone consistently. Despite some pitfalls in the initial design compared to the finished product, the deliverable itself still represents a project that has the possibility to create change in a real life scenario.

**Chapter 2. Background and Related Work**

* 1. **Background and Technologies**

Drones have been used by the military for decades and up until quite recently they haven’t been readily accessible to the public. Within the past decade we have seen a lot of innovation in the drone industry. We are beginning to see the various applications of drones not only for military use but for things such as agriculture, exploration of areas, and to help the community in ways that don’t endanger others’ lives; a search and rescue team could send an autonomous drone into a fire versus them risking their life to do the same job.

Several different militaries and police forces are incorporating drones and rovers into their arsenals to prevent various types of attacks. Most of these drones are used for crowd control or doing something that the police force wouldn’t otherwise be capable of. In Japan, they are employing drones that catch other drones in order to prevent them from continuing to fly in a no fly zone [16]. In India, the military uses a drone for riot controlling purposes because the drone can get to vantage points that police cannot [16]. Lastly, Cleveland’s police department is utilizing a rover in order to search and detect bombs [16]. What these drones and rovers have in common is that they are used to do things that police cannot, fly nimbly in tight areas, crawl under various structures, and fly in tight quarters. Lastly, their main objective is to replace what police do but better. By doing this, they protect the lives of police and do so at a fraction of the cost.

These drones present possible solutions for our needs but many of them do far more than what we need and their cost is far greater. Our drone requires far less features such as paintballs and tear gas, and realistically only needs video. Our drone isn’t necessarily what will actually be apprehending the criminal but what will aid in their arrest. Most drones on the market have cameras attached but many of these use proprietary software which we cannot work with. In order to create our drone, we need a drone that is open source and allows us to use the flight controller of our choice. Additionally, the drones with proprietary software do not have object avoidance as a feature and many lack the ability to program flights. If they do have that ability, they are often far too expensive for our application.

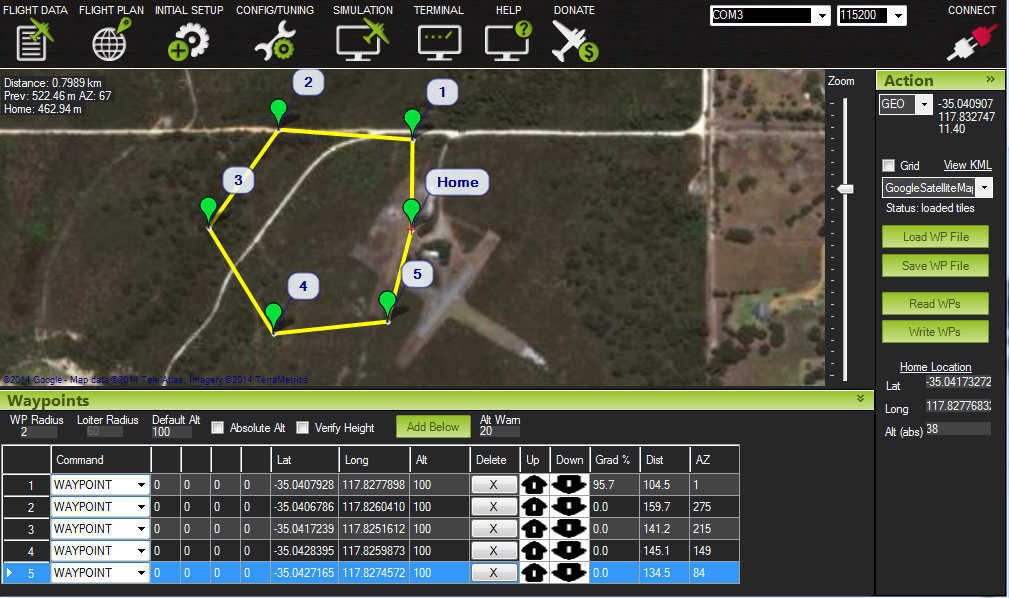
With all the drones having surveillance capabilities we have to have some way to control it and that’s where the software portion comes in. We have a mounted camera on the drone that records what is going on in the surroundings and that is being controlled by some type of software. This obviously comes into the category of surveillance which is why for certain drones it is required to register them with the FAA(Federal Aviation Agency) and in some cases you have to obtain certain certifications from them to operate the drones.

Drones and the software that has been uploaded to the hardware can serve different purposes. DroneDeploy is a popular software that allows people to process maps, interpret data and 3D modeling. With the amount of sensors that are available we can use the drone for many different purposes and have it gather and analyze different types of data.

The coding can get really complicated when dealing with the drone and the camera interacting together. There are currently multiple autopilot softwares that take care of that and allow us to make our own edits to it. The one that we are currently utilizing is the Crazyflie 2.0 and it provides several libraries with different functionalities. The code is mostly written in Python.

Almost all autonomous drones in the market right now implement some sort of flight controller software that fully utilizes the sensors onboard to allow the drone to be controlled autonomously. The most common flight controller that is open-source is Ardupilot, it provides multiple different functionalities for a variety of different uses. The autopilot software currently requires some embedded hardware that is compatible with what needs to be done and the most commonly used one with Ardupilot is the Pixhawk. The pixhawk has a 32-bit ARM cortex M4 core with FPU which is running at 168 Mhz/256 KB RAM with 2MB Flash ROM. It come with a variety of sensors that allow the drone to autonomously fly with respect to its environment. It has a MPU6000 accelerometer and gyroscope along with a ST Micro 14-bit compass/accelerometer which they call a magnetometer. The altitude is measured and monitored with a MEAS barometer.

The software that is used to control the drone is equally important and requires a lot of work if done from scratch. There are a lot of open-source flight controller software’s that are available the most common one being the mission planner with the Ardupilot. The features that it allows are the main reason it is widely implemented; it allows users to download mission log files and analyze them. You can interface with the PC flight simulator to create a full hardware loop, which basically is what allows the drone to fly autonomously. You can use any of the common communication methods to connect the mission planner to the autopilot software i.e. Bluetooth, radio, USB cables and IP connections. A basic layout of the mission planner is given in figure 1.



**Figure 2*.*** Mission Controller layout

Currently drones are used for many different purposes, there is a huge hobbyist market along with a fastly growing developer market. One of the main advantages of using autonomous drones are that you can program a specific function for it to complete and you won’t have to manually control it. This complicates things a bit as well. For manual control of a basic quadcopter as shown in Figure 1, you only need three basic controls, roll, yaw, and pitch. However, when entering the world of autonomous drones you need to know the position of your drone with respect to its environment; moreso the UAV needs to know where it is in 3D in order to make sure it can proceed.

The current method of autonomous control for the CrazyFlie 2.0 is using a “Flow Board” which is basically just an optical flow sensor and a ToF(Time of flight) sensor. The Flow deck consists of a VL53L0x ToF sensor and a PMW3901 optical flow sensor. The ToF sensor gets the distance to the ground with high precision and the optical flow sensor tracks the movement of the drone with relation to ground. The other part of getting the CrazyFlie 2.0 to fly autonomously is through the use of the location positioning nodes and location positioning deck that is available through bitcraze. The way it is currently works is the loco positioning deck communicates with the nodes and figures out the distance from the nodes and uses that to figure out its position. This works in a room but doesn’t work for the application that we were intending it to use.

**2.2 Literature Search**

The technology of drones has been around for decades, however, they are just beginning to see mainstream appeal. They have also been utilized by the military for years but were not very popular with the public. Since their gain in popularity, more studies have been conducted on and with drones and these findings can now be found in several scholarly journals. In a recent article from 2016 titled “Reactive Controls of Autonomous Drones” by Bregu et al., it describes the use of autopilot software and its implementation with the drone of hardware. This article is just one of the many articles which have been written on the subject of autonomous flight as well as drone flight. Some of these articles have been referenced throughout this paper and can be seen in the reference section of the workbook.

Using the work of Saska et al., we were able to discover the possibilities of utilizing drones for autonomous flight [17]. Their work shows that a drone can be capable of navigating various environments and is very robust in its recognition and detection of objects. Their research paper was extremely relevant to our scenario, especially where they highlighted the fact that the use of the drone is more advantageous to the use of a drone which is similar to our application. They also highlighted the ability of their drone to interact with a rover in order to operate in a synergistic manner which represents a possible use for future work relating to this project.

Tang et al. and their work specialized on the use of drones for research relating to deforestation in jungle like areas [19]. They deployed their drones with the desire for them to fly autonomously and use remote sensors to determine canopy height relative to the height of the drone. We can use these findings to advance our own development by choosing relevant sensor technologies available. Their research also demonstrates the benefits of using the drones compared to a crew. In the instance of tracking a wildfire, the drone could get closer to the fire than a crew would ever chance and it could easily track the fire. The use of a drone provides greater safety compared to a crew flying near the fire. They did also remark about the potential risks when using a drone compared to a crew, highlighting the facts that drones are susceptible to weather related accidents and accidents related to human error.

Work by Lugo and Zell was most relevant to our project [18]. Their experiments took place in 2013 so their tools may be dated, but their methodologies are still relevant. Many of their approaches and the platform they used was similar to that of ours. Their drone had onboard sensor processing as well as utilizing a GPS module for navigation. Their experiments took place in two separate rooms, however, where our desired testing would be outside in a cityscape. Their findings and experimentations did provide a great starting point for us to reference. Their work provided us insight on how to design our drone architecture and the details of their work allowed us to plan for future difficulties. Seeing how the additional weight of a microcontroller affected their drone’s ability to fly effectively, we knew that we would need to make ample modifications to generate greater speed and flight time.

The drone that we have decided to implement the autonomous software with is the Crazyflie 2.0. This drone is the successor of the Crazyflie 1.0 created by Bitcraze. Because it is open source and very new, there are little journal articles dedicated to this particular drone, however, there is a plethora of articles and forum posts dedicated to this drone that can be found online and are referenced below in the works cited. Lastly, the autopilot software to be utilized, PX4, is an open source software which has large amounts of documentation, and is referenced in the works cited section of this document.

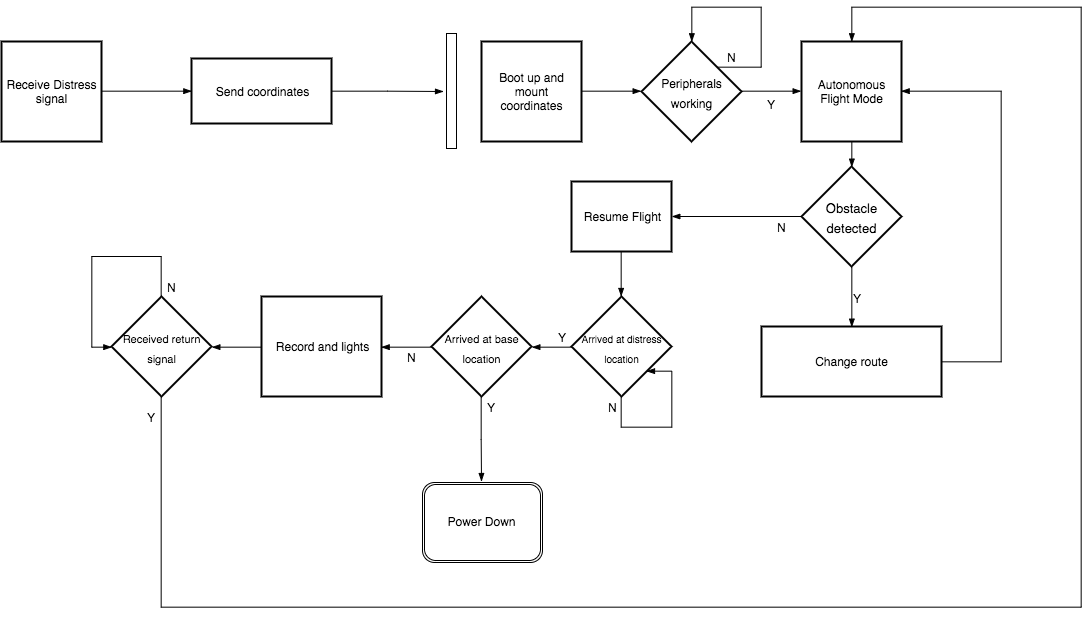
**2.3 State-of-the-art**

The drone market is flooded with drones specific to the the individual needs of the person purchasing the drone. The drone is growing at an exponential rate, according to an article on record global sales grew about 60 percent in the last year, with the hobbyist market almost doubling. Drones are becoming more compatible and some can even be controlled through the Android platform [14]. The leading company in the hobbyist community is definitely DJI which make drones with excellent video and photo functionalities. The current drone they offer which is considered the best in the field is the DJI Mavic Pro which can go up to 65 mph and has 27 minutes of fly time. It has a range of about 7 km and has a 3-axis gimbal attached to the camera. Currently we are looking into editing and making changes to their drones if possible. In terms of having a stable drone that we can use to perform the functionality we need, it would be better to try and “hack” their software and make the necessary modifications.

There is also a growing market for drone racing, it’s to the point where people are talking about drone racing being an actual sport. Most drones outside of racing racing drone are using LiPo(Lithium Polymer) batteries for the following reasons: low profile and rechargeable, sizeable charge density, and with high C ratings can deliver lots of current without too much degradation. A pretty good racing drone that we examined was the KingKong 210GT. The components used on this drone were something that we considered, however we later decided not to pursue this drone due to the controller software being incompatible with our expertise. The 210GT runs the F3 flight controller, which is a suitable product; however, we wanted something that would be more open source and developer friendly where we could make some modifications.

# **Chapter 3 Project Requirements**

**3.1 Domain and Business Requirements**

****

**Figure 3.** UML2 Activity diagram demonstrating state transition

As shown in the UML2 activity diagram, the drone begins with the base receiving an input from the “distress signal” and then transfers the desired GPS location to the drone causing the boot up procedure. The drone check that the peripherals are working and then proceed to flight. Once the drone is in flight, it goes towards the destination and continuously checks for an object. If there is an object in the way, the drone changes it route and resumes its original protocol for flying towards the location but still checks for objects in the way. It runs a check to see if it has arrived at the distress location and once it has arrived, it checks the flag for having arrived. This flag will be used later for checking if the drone has arrived at the base station location. Once it’s at the location, it begins to record and illuminate the location of the distress signal. After it receives the return signal, it begins its protocol

**3.2 System Functional Requirements**

Within the system there are several functional requirements that are required to successfully build the MVP. The functional requirements are the requirements deemed necessary to complete a fully operating system. There are requirements that are considered non-functional requirements which are requirements that describe the numerical performance of the system.

***Functional Requirements***

* The system shall be able to turn on from the base station
* The system shall fly to the desired location of the user
* The system shall utilize the GPS module
* The system shall avoid objects during flight
* The system shall begin recording when it reaches the desired location
* The system shall fly back to the base when the return signal is sent
* The system shall go into a wait state for the next signal

**3.3 Non-functional Requirements**

* The system shall take no longer than 30 seconds to boot and take flight
* The system shall maintain a flight time of at least 15 minutes
* The system shall carry the payload of the Raspberry Pi and other peripherals
* The system shall go to the correct destination 90% of the time
* The system shall be capable to operate at a range of at least ½ a mile
* The system shall take at least 5 minutes of video

**3.4** **Context and Interface Requirements**

The development was comprised of two sections: hardware and software. The hardware was comprised of several different modules which came together to form one larger system. The CrazyFlie 2.0 controller was mounted on the Xiro drone frame which had a new ESC for generating greater power. The ESC was powered by a new Lipo battery and the Raspberry Pi module as well as GPS module were mounted upon the drone.

Regarding the software, the development came from various locations of open sourced code. The CrazyFlie drone is an open source drone, so many solutions are available. The software would be flashed on the Crazyflie over radio signal. The Raspberry Pi also had to have code mounted on it to implement camera recording or photography applications.

The testing for the modules could sometimes be conducted separately. For instance, the testing and verification of the Raspberry Pi module could be completed outside the context of the system. For the GPS, it had to be tested on the system in order to generate the readings needed for debugging. The frame could be tested first by adding the new ESC and battery and testing the drone manually for testing the payload and power generated. The Crazyflie software could be tested by mounting it on the Crazyflie first and the putting it on the Xiro drone frame.

The deployment of this system took shape in the form of the fully functioning autonomous response drone system. It is a full drone system which incorporates each module into one functioning unit. It has the ability to take an input signal and travel to that location and avoid objects on the way. Additionally it has the ability to record video and maintain stable flight. Lastly, it can return to the base where it can be charged again and waits for the next signal to be issued.

**3.5** **Technology and Resource Requirements**

| Part | Description |
| --- | --- |
| XIRO Xplorer Aerial UAV Drone Quadcopter | Drone Chassis and motor assembly. |
| RacerStar 4-in-1 20A Brushless ESC 2-4S for Quadcopter Racing Drone | Electronic Speed Control. |
| SD-card deck | N/A |
| Buzzer deck | N/A |
| BigQuad deck | Expansion board for CrazyFlie. |
| CrazyFlie 2.0 | Small open source drone. |
| 6-cell AA battery holder | Battery power for smaller components. |
| Molex Connectors | N/A |
| Molex Contacts | N/A |
| LiPO Battery | Battery to power bigger motors. |
| 4 x IR Sensor | IR sensors for various uses. |
| 4 x 3 pin headers for expansion board | N/A |
| uBLOX MAX-M8C Pico Breakout with Chip Antenna | GPS module with custom firmware. |
| 1 x telemetry module | Addon |
| 1 x radio antenna dongle | Addon |
| 4 x 3 pin molex connectors (3.16mm or .1inch pitch) 2.65mm or .1 inch pin width | N/A |
| 6 x Energizer AA batteries (ENERGIZER NH15-2300) | Batteries for battery holder. |
| 1 x 4 pack AA battery cell holder | Battery power for smaller components. |
| 4 x Break-away 0.1" 2x20-pin Strip Dual Male Header (PRODUCT ID: 2822) | N/A |
| 4 x GPIO Header for Raspberry Pi A+/B+/Pi 2/Pi 3 - 2x20 Female Header (PRODUCT ID: 2222) | N/A |

**Table 1.** Parts list and description

The software stack is split into two main parts, one handles the drone and other external peripherals. The drone consists of a bootloader which is flashed onto the drone for extended onboard functionalities. While the firmware allows external boards and custom designs to be produced. Uploading the crazyflie custom ground control firmware and bootloader these open source software libraries were used. The libraries are mainly low level C code with a simple wrapper of python. Since we are integrating custom hardware the firmware and bootloader are built for a specific set of hardware.

The second part of the code deals with the RPI running on linux based debian systems. The goal is to be able to store video and audio data on microsd card code and be able to upload it to a server. This will include battery management, elevation, audio, video, led control, and all internals.

**Chapter 4. System Design**

**4.1 Architecture Design**

ARD is designed around the concepts of stability, simple assembly, and modular simple maintenance. The chassis is a *XIRO drone* frame which features a quad motor configuration with blade style propellers, and a retractable quad leg landing gear system. The chassis is able to house the several primary components: Crazyflie 2.0 chassis and microcontroller, BigQuad expansion deck, electronic speed controller (ESC), and the situational record unit (SRU). The overall chassis is made of formed plastic, and houses several built in components: 4 DC brushless motors, LED’s, and system wiring for easy integration.

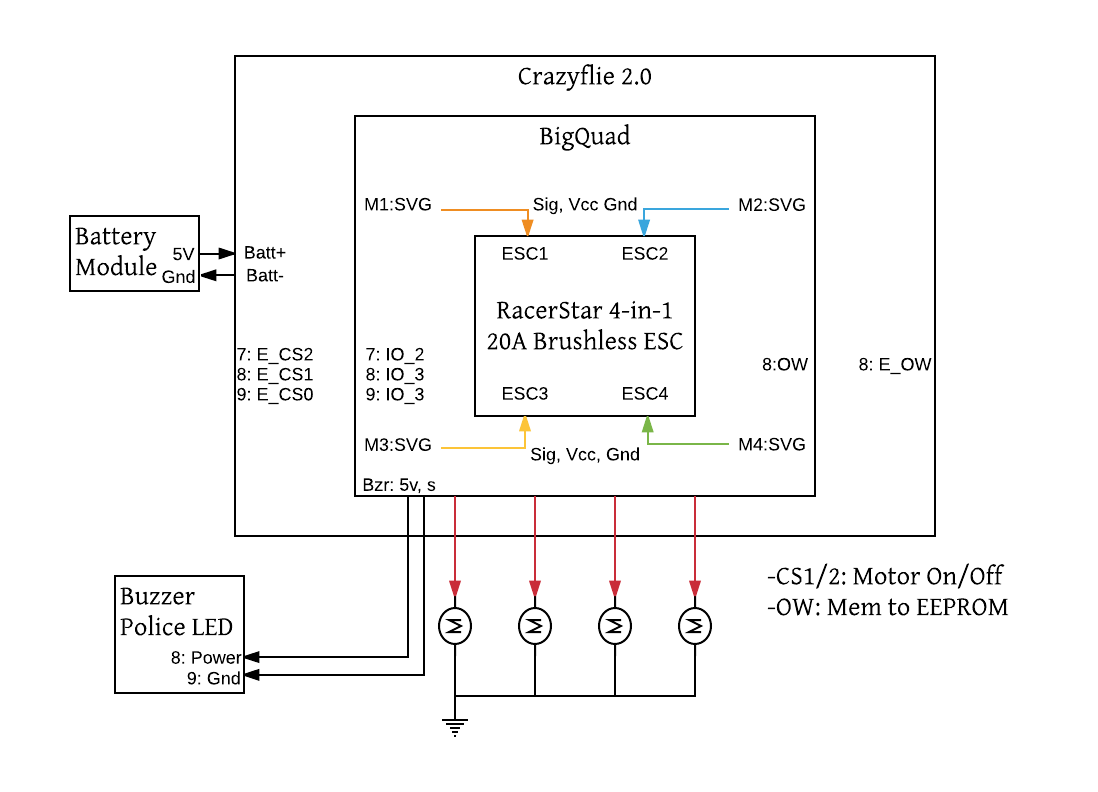
The core of ARD is the SCB, which is a Crazyflie 2.0 micro controller built around the STM32F405xx processor, which outputs PWM signals to the ESC for motor control. This board also contains several components required for aerial embedded systems: pressure transducer for altitude sensing, 3-axis accelerometer for tilt angle sensing, 3-axis gyro for rate sensing, and a 3-axis digital compass which helps the drone find its heading with respect to true north. It also uses the nRF51822 radio and power management MCU which is a Cortex-M0 architecture. This board performs a dual purpose as it is used for power management as well.

The ESC is a 4-channel DC-AC controller. It takes in 4 separate PWM signals from the SCB (one for each motor), and distributes the signals to their addressed motor. It utilizes a 11.1V DC power from a 2200mAh Li-Po battery that is rated at 20C. The ESC can handle a distribution of up to 20A across for each AC Brushless motor, but runs at a constant 5A per motor.

The motors are each rated for 100W, and take in 3-phase AC. They are axial-flux motors which means their outer housing rotates as the electromotive force is applied to it. This means the motors are efficient as compared to DC brushless.

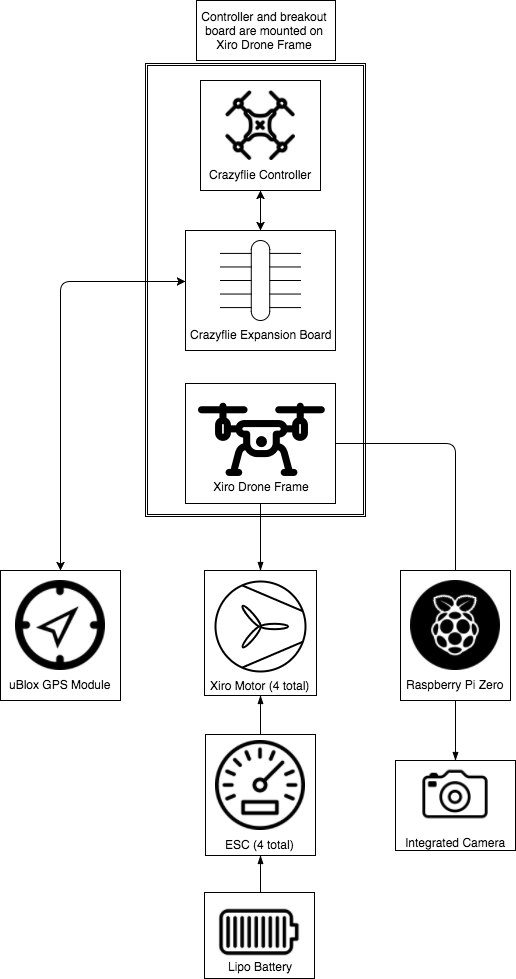
All electronics with the exception of the high powered components (motors and servos), will be powered by a filtered and regulated 240 mAh LiPo battery. This will allow the engineers to power any onboard device via their respective power port (e.g. USB port, through-hole header, screw terminal fastener, etc). The board offers simplicity in terms of power management and application to ensure long lasting life.

For the user interface, there will simply be a handheld device such as a mobile phone with an application where the user can just send a request to the main station, and that will in turn activate the drone. It maintains a constant broadcast for the drone to map its way to the user in distress. Once the range has been minimized, the drone will then know the location of the user, as be able to capture the on scene information.



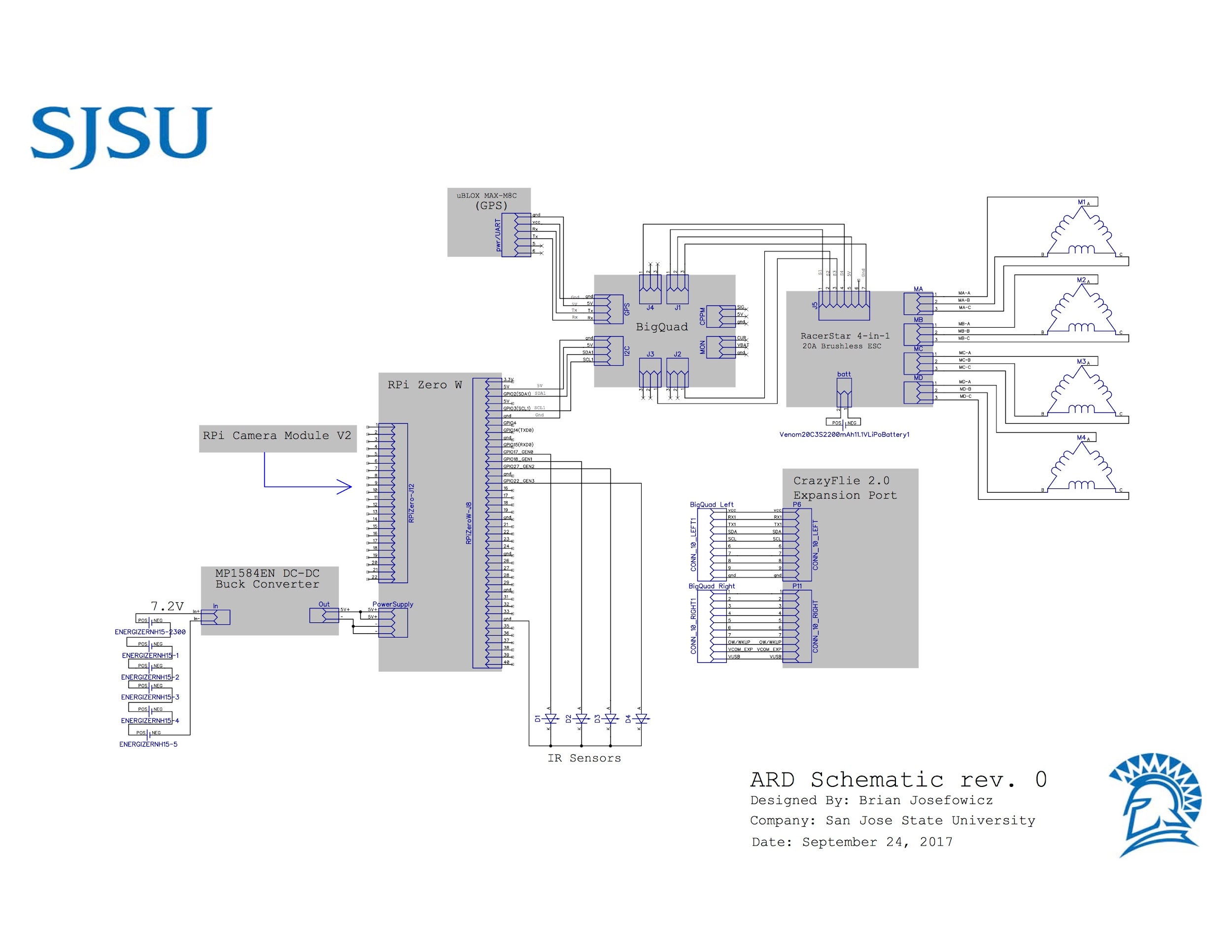
**Figure 4.** ARD Central Stack. The ESCs, which control the motors, are connected to the BigQuad pins. The BigQuad is connected to and interfaced by the CrazyFlie controller. The controller is powered by a 5V battery module.

**4.2** **Interface and Component Design**



**Figure 5.** Overall system block diagram. The “main stack” contains the CrazyFlie drone controller, the BigQuad Expansion board, and is all mounted atop the Xiro drone frame. Attached to the Expansion board is the uBlox GPS module. The Raspberry Pi Zero is mounted on the Xiro drone frame and interfaces the camera. The Xiro motors are controlled by the ESCs which are controlled by the CrazyFlie controller. All of these devices are power by the lipo battery.

**4.3** **Structure and Logic Design**

****

**Figure 6.** Diptrace pinout of full system. Pinout includes the Raspberry Pi zero which can be seen interfaced with the camera module and a DC converter and IR sensors. The expansion port shows the pinouts that go to each half of the BigQuad expansion board. The BigQuad expansion board interfaces with the Raspberry Pi Zero, the GPS module, and the ESCs. The ESCs are interfaced with the four motors.

**4.4 Design Constraints, Problems, Trade-offs, and Solutions**

Throughout the process of planning and development, engineers must foresee the possible setbacks in order to mitigate risk, and have a chance to design around the issues that can arise from unforeseen failures. Certain parameters play a key role in creating the proper design plan going forward, and here are the factors that were considered during the planning and design process.

* 1. **4.4.1 Design Constraints and Challenges**

The primary design constraint is of course time. We need to be sure that our design fits our capability to deliver a working product on time including testing, regression, and possible changes that may occur. Similarly, we are limited on financial contribution which means we need to not only build a reliable product, but a cost effective one too. Individual mistakes turn into sunk costs, so it is a goal to minimize the reality of these risks.

A large constraint is ensuring proper integration between hardware and software. In our design, we will be modifying the drone to take on a larger chassis. With these modifications, unforeseen circumstances may arise. The drone may not be able to generate enough power with the new chassis or the drone’s balance may be thrown off. These types of challenges have to deal with and overcome either with additional hardware modifications or modifications to the software. In order to have a successful autonomous drone, the drone’s hardware must be able to work effectively with the software. The drone must properly relay information from the sensors, and the code we implement must adjust accordingly.

While there are some hardware changes being performed on the drone, those challenges are miniscule compared to the challenge of programming for autonomous flight. The drone will largely act like a state-machine, following various states to achieve takeoff, flight, and landing, however, these steps themselves are very intricate. In order to execute autonomous flight, the drone must be able to recognize the location of a signal and properly reach that destination. This includes using its own sensors to avoid obstacles, and that can be implemented with sensors that mimic vision for the drone, or exteroceptive sensors [6]. Using IR or laser sensors, the drone will be able to accurately tell the distance from an object and following its programming, be able to avoid it and continue. The drone will also have to ensure that its in proper working order to be able to make the flight. Damage to certain sensors or a low battery would hinder the success of the drone flying to its destination. The drone will make use of proprioceptive sensors which monitor the drone’s internal status [6]. This will ensure safe flight and prevent damage to the drone or to any objects or people.

**4.4.2 Design Solutions and Trade-offs**

To compensate for the low load capacity of the CrazyFlie, an additional chassis and ESC were purchased. These additional components will allow for us to carry a heavier payload and provide more stable flight. With this additional chassis comes extra weight. The tradeoff of being able to generate more power is that the drone will consume more power from the battery. With this in mind, a shorter flight time and more consistent charges will be required, or the purchase of a larger battery may be necessitated.

The use of external, third-party sensors is a solution we came to in order to provide a more accurate and controllable object detection system. Compared to other drones which are equipped with cameras, the sensors will work more effectively and do not necessitate creating an object detection algorithm based on a 2D image. The drawback of using third-party sensors compared to the proprietary components is that they may be harder to manage. While it may not be simple plug and play like the cameras that come standard on some drones, the CrazyFlie’s expansion connectors and implementation of the big quad deck will allow for simple integration of various third-party sensors and components

**Chapter 5. System Implementation**

**5.1 Implementation Overview**

The very high level design as described in previous sections is to build an autonomous drone with obstacle detection in order to cater to the security needs of students on campus. With the CrazyFlie 2.0 being an open-source and developer friendly drone we were able to customize it to the specific requirements that we needed to meet. There are many different platforms that we had to use and incorporate with this project. Starting with the CrazyFlie 2.0, the open source code for the PX4 Pro Autopilot software is all available on github and is mainly written in C and Python with some python packages that needed to be installed in order to successfully flash the drone in a macOS environment. There are many other CrazyFlie specific platforms that we used in the process of getting familiar with the drone some of which are, the CrazyFlie Python Client which basically gives us the same interface and functionality as the CrazyFlie Client software that is used in the Bit Craze VM which we also used. The CrazyFlie PC client is all written in Python and is still in somewhat of a beta stage so we had test it and incorporate other tools as well to get all the functionality that we needed.. The CrazyFlie client allows us to flash and control the drone in a mac OS environment rather than using the Windows version or the Bit Craze VM; which made some things easier since most of our development was being done in a Mac OS environment.

The next platform that was a crucial part of the development was the mobile application that was iOS based. The basic functionality was to control the drone using our phone and this was proven to be a crucial step throughout since we had to test the drone after any software/hardware changes were made. The radio/MAVlink was used to communicate with the drone as well and all the firmware for that was written in C/Python, this part was easily incorporated into the project since it was just a couple extra files that needed to be added into the right directory. For the hardware side of the CrazyFlie 2.0 we had a few different things that needed to be addressed, we have two different microcontrollers being used. The ARM cortex-4 microprocessor is what controls the main applications that being run and the Cortex-0 microprocessor is what handles the power management and the radio controls. We didn’t have much development to do in terms of the CrazyFlie 2.0 hardware, the PX4 Pro autopilot software was written to be compatible with the existing CrazyFlie 2.0 hardware. Most of the changes that were made to the software were all then flashed to the CrazyFlie 2.0, since we had added the BigQuad deck to the CrazyFlie 2.0 we had to make the appropriate changes in the source code to enable that specific flag this was done using a simple text editor Sublime in Python/C and then added to the appropriate directory.

The PX4 Pro autopilot software comes with its own native mission planner software/ platform. The QGC mission planner is ground control software that allows the user to send the drone on autonomous missions; this functionality can only be achieved with the addition of a GPS module to the drone since the software needs to be able to locate the drone and know where it is continuously to make sure it’s on the appropriate path. The program itself is built with a very easy to use UI and all it needs is for the on-board sensors to be properly calibrated for the drone to work with the QGC. The uBlox M8 GPS was needed in order for the PX4 Pro Autopilot software to be able to locate the drone, without GPS location we can only fly the drone in manual control mode which is a fail safe in case something goes wrong. The GPS we used needed the firmware drivers that allow us to use the sensor with the PX4; this was written in C/Python and added to the PX4 project.

The CrazyFlie 2.0 development was done independent of the Raspberry Pi system. The main function of the Raspberry Pi in our project was to be the eyes of the drone. This was done by mounting the camera with the raspberry Pi on top of the drone and having it record and save everything on a microSD card on-board the Raspberry Pi. The platform that we used for this was the native Raspberry Pi Desktop which is a Debian (Linux) based Operating System. This operating system comes with a lot of pre-installed programs made for developers. The code for the camera was all written in Python and uses the Raspberry Pi operating system, we save the recorded file on the microSD card and then pull the file using the command line interface and using SSH to access everything on the Raspberry Pi.

The R-Pi was also interfaced with our laser sensors that are used for obstacle avoidance/ detection. The platform was the same as the camera, the standard R-Pi debian based operating system. The ping sensors easily interface with the R-Pi GPIO pins which allows us to use them for the functionality that we need. After trying to use the Eclipse IDE we found it be easier to simply make any changes in Sublime Text Editor, compile and test which is what we ended up doing. Most of the sensors such including the one we chose have a lot of open source code available online; adding to the firmware drivers that were found online we were able to successfully interface the sensor to the R-Pi Zero.

The Xiro Xplorer drone that we used was a ready-to-go drone out of the box that came with many different functionalities. The basic out of the box specs include a 5.8G data transfer speed which allows for high quality video streaming. It comes with a LiPo 3S 1650mAh battery that allows for about 23 minutes of flight time with a maximum flight height of up to 300m. The communication is supported on a 5GHz channel that would allow us to control the drone remotely. Our first approach was to use to the same microcontroller and make changes to the software with the help of the Xiro Assistant software. The software is almost similar to the QGC ground control station however the issue with this platform was that it was all in Chinese which doesn’t provide us with much help. After a lot of testing it was deemed not feasible to make edits to the existing setup therefore we chose to proceed with the CrazyFlie.

The design involved powering the Crazyflie 2.0 module with an external battery that would be used to power the whole system. The default setup came with a LiPo (Lithium-Polymer) battery specifically the 671723HS25C that provided the micro controller 3.7V at 170 mAh. The original battery also had a PCM (Protection circuit module) that prevented the battery from being shorted, under-charged or overcharged this functionality was proven to be crucial after we had shorted a few batteries. Considering the weight of the battery only being 3.0 grams we decided to keep the original battery to power the CrazyFlie 2.0 battery. The next step was to power the BigQuad expansion deck. The BigQuad deck inputs are all 5V tolerant and get connected to the expansion headers of the CrazyFlie 2.0. The expansion headers of the CrazyFlie serve many functions that get extended to the BigQuad; the pin functions supported are SPI, ADC, UART, I2C and power, of which 4 pins are GPIO.

Once the design for the power management system was successfully created and tested we had to approach the communication design. The CrazyFlie 2.0 comes pre-built with an on board IMU (Inertial Measurement Unit) and AHRS (Attitude and Heading Reference System) unit. The IMU is configured with a 3-axis Gyroscope (MPU-9250), 3-axis accelerometer (MPU-9250), 3-axis magnetometer and a high precision pressure sensor (LPS25H). This whole system is interfaced with the MCU (Microcontroller Unit); with the help of a Saleae logic analyzer we were able to determine that the communication was via I2C. The first approach we took for interfacing the camera and the laser sensor was to add it to this subsystem however that was proven to to unsuccessful due to many reasons that will be discussed in the later sections. The onboard radio module was tested at 2.4 GHz and using the logic analyzer we found that it communicated to the MCU via SPI protocol. Luckily, the 2.4 GHz allowed for BLE (Bluetooth Low Energy) which was a crucial part of the testing process since the BLE allows for us to use the iOS application to control the drone.

**Dependencies - can also be found in the Appendix**

The Crazyflie has the following dependencies:

* Installed from system packages
  + Python 3.4+
  + PyQt5
  + A pyusb backend: libusb 0.X/1.X
* Installed from PyPI using PIP:
  + cflib
  + PyUSB
  + PyQtGraph
  + ZMQ
  + Appdirs
* SDCC
* Binutils (needs objcopy)

**5.2 Implementation of Developed Solutions**

The drone core comes in several different pieces: CrazyFlie 2.0 drone, BigQuad expansion board, and the 4-channel ESC. By taking each component and its constituent hardware, it is intuitive to follow through with the mounting process. The interface should mate the CrazyFlie 2.0 to the top of the BigQuad board. This will allow all control signals, and data communication to flow through the core to peripherals. More specifically, the control signals needed were 5V supplied to the added boards, UART RX and TX, I2C SDA and SCL, OW/WKUP, and the VCOM\_EXP pin. The 5V supply was needed for powering the peripheral boards. This was important to note because later challenges were faced with powering the ESC. UART, RX and TX were needed for taking in data from the GPS module. This needed to be routed the BigQuad board for two reasons, and will be mentioned in the following challenges section. The I2C pins were needed for receiving data from the RPi boards. This data contained the peripheral distance values that are used for object detection. OW/WKUP is the proprietary BitCraze Expansion board memory power on pin. It is essentially used as a chip select via indirect memory mapping. If a value is seen at this pin, the CrazyFlie knows there is a device present and will adapt its software for sending it signals. The VCOM\_EXP pin is important for also communicating with the expansion boards. The pins listed here all operate a 5V logic and are interfaced directly with the BigQuad Expansion.

The BigQuad expansion has several interfaces needed for our design. To elaborate, GPS, I2C, and motor control were interfaced through this board. As mentioned, the signals were interfaced with the drone via its two headers. The peripheral interface however was done via three or four individual header pins. Some challenges with this are described in the following section, but the main concept was successfully verified in testing. Each motor connector only required one wire with the exception of J1. This connector needed to take in not only PWM, but 5V and Gnd as well. Then mounting the 4-channel ESC to the bottom of the BigQuad gives accessibility for the PWM signal pins to be connected between ESC inputs and BigQuad outputs. These signal pins and constituent 5V and Gnd pins provide power, gnd, and motor signal control to the ESC. This setup is what makes up the entirety of the drone core. The ESC 20ga wires were then appended with Molex mini-fit junior connectors such that they can be connected to the chassis header connectors.

Peripherals to the drone core include: two Raspberry Pi Zero W boards with attached HD cameras capable of delivering 1080P video feed to the ground control station, a GPS module that provides accurate real-time GPS location of the drone, four Arduino laser sensors attached to the RPi GPIO pins providing distance to object recognition, and a low current battery power supply regulator board eBoot Mini MP1584EN DC-DC Buck Converter. The interface between Raspberry Pi Zero W boards and the drone core will be handled using the I2C communication protocol. The implementation will allow us to send messages from the RPi to the drone core registers which will store data for the PX4 autopilot software to read from. It is imperative to ensure that all firmware developed for the RPi’s have the proper addressing references for the drone to operate properly.

The laser sensor components allow us to sense the response time between laser output, and reflected signal input to the RPi. This time data will be stored into 4 separate bytes equating to a 32-bit value for interpretation by the PX4 firmware. The 32-bit value is sent over the I2C bus, and received by the PX4 control system for auto piloting the drone in an avoidance pattern pattern. We have also developed an object avoidance redundant system such that the PX4 control system fails. The concept is to stop the drone when an object is detected within 15 meters. If the object location persists, the drone should raise its elevation until the object is cleared, or another object is detected. If the drone cannot clear the object within a 10 foot elevation climb, it should then begin sending calculated GPS coordinates using a heading reference. While maintaining the original GPS location, the drone will essentially be navigated around the object until there is no longer an object impeding the line of travel towards the initial location.

To mount the core, high current battery, and low current batteries along with the surrounding peripherals, we needed to design a rapid prototyping mounting fixture as seen Appendix figure 1, 2 and 3. This allows us to place components on the frame without offsetting its geometric weight balance. Our hardware also reduces the overall weight of the drone allowing us more payload room for adding the additional peripherals. Although our high current battery possess half the capacity of the OEM part, we have a smaller profile package, and a 63% weight reduction. The battery also meets our ESC’s 20C rating. The low current 6 AA cell pack provides the necessary current and voltage to supply power to the RPi boards and cameras.

**5.3 Implementation Problems, Challenges, and Lesson Learned**

Implementation problems mainly revolved around proprietary hardware and software pairing with open source software. Finding the proper drone configuration was a lengthy step as multiple drone options were considered. One option considered was a drone that already came with a camera which would solve having to interface an external camera. There were some problems with this option. One problem would be the object avoidance. Because the drone didn’t have other sensors or a way to interface them, a program would have to be developed which would detect objects based on the filled color ratio from the camera feed. This would end up being a problem if one were to choose to use the drone at night. Another problem was that the drone was based off proprietary software. This meant we would have to hack the drone to get it to work or to flash it with an open source software, both of which would be more work than it’s worth.

Problems with weight, power, expansion options were our limiting factors to the project. Initially, the drone in use would solely be the CrazyFlie with software that enabled it to hover. We later found out that the payload which had the peripherals we wanted to implement would be too heavy for the drone. Additionally, our flight time would be limited even if we had the payload maxed out which would not be satisfactory. To deal with this problem, we decided to implement an alternative drone as a frame in order to utilize its size and power. This drone made it possible for us to carry our desired payload and a larger battery pack for extended flight. We were able to take apart the proprietary ESCs and install our own. By doing this, we were able to generate more power in order to carry the additional payload.

Challenges were also faced during software development. The main challenges were the compilation and flashing of a custom bootloader and custom firmware upon the drone. Most of the drone software and hardware is still in beta which meant that there were few resources for us to use. This meant that research and testing took more time than we had estimated and slowed development. Additionally, the software that was used was only available for Windows users and a majority of the group used Apple PCs. This meant development could only be done with one computer which made it harder for us to collaborate. Additionally, the BigQuad expansion board is still under beta testing, along with the PX4 firmware for the CrazyFlie. With the expansion board and the PX4 firmware still in beta, we ran into problems with programming the controller. It was challenging for us to debug and find the errors we had because there were so few resources available to us. The documentation, while helpful, was not sufficient. The documentation helped us to develop and find solutions but it didn’t help us to figure out our problems. On occasion, we were able to find help on the BitCraze or PX4 forums but most of the time it was up to us to resolve the problem.

Overcoming challenges has pushed the project forward. The biggest lesson learned is to pay attention to every detail and research part and product before assuming it can be implemented. We were often overconfident in our ability to implement each peripheral and it made development challenging. We would try to implement the module we chose when there may have been a more suitable option available. This was a problem when choosing the sensors and GPS module. If we had researched all of the peripherals prior to developing the project, the integration of the peripherals would have been much easier. Researching the payload of the drone and looking at other options may have also saved us time, time which we could have used for further testing. By buying an additional frame, ESCs, and battery, we spent more money on a “cheap” drone than possibly buying a costlier drone with all of the desired peripherals.

During the project assembly and testing phase we ran into several compatibility and quality issues. The first step was to remove the proprietary Xiro drone core, and then hook up a set of Molex mini-fit three pin connectors from the ESC. Although we looked closely into the header pitch and pin width, mating the connectors with the proprietary headers was not possible due to pin incompatibility. To accomplish this task, several methods were attempted. The first option was to expand the pins inside the Molex connectors, and shove them over the larger pins. This would not be optimal as it could damage the smaller pins, and worse allow us to pass a higher current through a smaller pin which could then lead to overheating and damage of the drone. Second, we needed to create an impromptu interface by soldering wires to the headers, and protecting them with an enclosure filled with RTV. This was our best option because it allowed us to extend the connection points to well outside the exterior of the drone, and for us to add any mating connector to the end such that we can connect our ESC connectors. This was a lengthy challenge to overcome; however, the improvements to interface quality were worth the time and effort.

Furthermore, the chassis mount design posed several challenges: size, shape, alignment, and space. The proprietary drone core was extracted and measured using digital calipers to allow precise dimensional analysis of the hardware. These values were recorded in the CAD project where the mount was being designed. The shape of the mount was the most important factor and challenge to meet in this process. Since the mounting area has an ‘x’ shape, each edge and contour had to be designed using the CAD tool. Numerous chamfers, and fillets were made on the corners, and creases to allow round shaped edges to fit securely inside the drone chassis. Alignment and space mainly affected each other as where the mounting holes for the units needed to be placed such that all of the hardware could fit. This was challenging because the space in the mounting area is relatively slim compared to the part dimensions. Several considerations were made here. Offsets were placed to allow some parts i.e. the RPi boards, and core mounting holes could fit side by side. The only way this would be possible was to consider using standoffs allow the edges the boards to overlap the mounting screw extrudes on the bottom. This was still possible as the battery could then be zip tied in between all of the units. Lastly, the mounting holes needed to be accurate to ensure proper screw insertion. Several iterations of the measurements were taken to ensure each screw hole was placed in the desired area within a tolerance of +/- 0.1mm. The depth of the screw wells were also considered to ensure the screws reached there mating holes. Overall, these challenges resulted in a successful mating of the mount and chassis.

To further elaborate on the manufacturing and assembly of the chassis mount, a couple of key challenges were faced. First, 3D printing of the chassis mount failed several times in early stages. The build up material of the mount edges continuously failed until it was realized that there was no need for the build up material if a < 45 degree edge was added between the outter lip, and the base of the mount. This was added, and the print ran successfully. Second, the room for hand access to the inside of the mount was limited. Routing wires, installing screws, and adjusting the hardware within made was made difficult by the limited space. This caused a couple of assembly iterations as proper alignment and fitment needed to be achieved. There were extruded cuts made in the mount design to allow connectors and wires to pass through; however, the size of the connectors was not considered, so the assembly became gated by whether or not the drone firmware compatibilities were ironed out first before adding the connectors. To clarify, the step by step process was as follows: CrazyFlie firmware verification, addition of the BigQuad and ESC, activation of the added units, assembly of the drone core with wires passed through, add the connectors (once installed, could no longer be removed), lastly mounting of the hardware. This serialized development abilities, and made firmware testing challenging as everything needed to be verified and fully tested prior to complete assembly. One solution for this was to loop the wires and connectors around the chassis exterior during testing for easy configurations, and once all tests were complete, remove and install a new set of connectors once the wires were finally passed through.

The presentation cannot contain a demo as San Jose city is a no fly zone, however testing and demo will be done at a more suitable environment. Most of the testing was conducted off campus, however, we conducted some very light testing on the San Jose State campus. Obtaining the fly zone license early on in the project, we were able to perform flight testing legally, but there were still limitations the hindered some tests. Traffic in streets made testing around roads nearly impossible unless the test performed at night. The GPS program built into the QGC does not consider these limitations, so we had to ensure that the drone could navigate freely without the prospect of running into traffic. Furthermore, campus buildings were an issue when navigating around the campus. The object avoidance would allow building detection, but this feature was not ideal as the drone routinely had to make large adjustments often after it had already proceeded to close to an object. The buildings also distort the GPS connectivity which would sometimes cause the drone to lose its way when in crowded areas.

**Chapter 6 Tools and Standards**

**6.1. Tools Used**

This project was comprised of both hardware and software tools. For the hardware, we utilized the CrazyFlie 2.0 drone. This drone was chosen due to its ability to utilize the PX4 open source software and can utilize a breakout board which enables the controller to integrate with several other peripherals. Unfortunately, the CrazyFlie 2.0’s lift proved to not be powerful enough for our application so we developed a system capable of providing enough lift for the controller and additional peripheral devices. To create this system, we used an additional drone frame which was larger and had larger propellers. We still needed to provide more power for this system to provide the speed we would like. To accomplish this, we added a more powerful LiPo battery and ESCs which could cause the motor on each propeller to operate faster and thus provide more lift. The last hardware tool we used was a GPS module which can interface with the CrazyFlie through the I2C pins on the CrazyFlie and can be utilized by the drone controller software.

There were two components which required software programming on this system. The CrazyFlie controller and the Raspberry Pi Zero both required software programming to utilize each of their functionalities. The CrazyFlie could be programmed through its PX4 firmware. We didn’t directly program the drone through its firmware, however, we utilized QGroundControl which provides a user interface for programming the drone. This user interface makes the project more user friendly from a programming side and it’s more powerful for testing. The Pi Zero also requires programming in order to utilize the attached camera and receiving a signal from the CrazyFlie controller. The Pi Zero will use multiple software protocols such as SPI, GPIO, and UART to control many peripherals. Software used for each external device comes with open source libraries. The leds, camera, and mic devices use Python and C libraries. With these devices being programmed, they were able to interact with each other and performed the actions we designed them for.

*Source Code Repository*

<https://github.com/s3nu/ARD>

*Open Source Disclaimer*

<https://git.drogon.net>

<https://pypi.python.org/pypi/RPi.GPIO>

<https://picamera.readthedocs.io/en/release-1.13/>

<http://archive.raspbian.org>

<https://github.com/raspberrypi>

<https://github.com/PX4/Bootloader>

<https://github.com/PX4/Firmware>

<https://dev.px4.io/en/flight_controller/crazyflie2.html>

**6.2. Standards**

This team adhered to strict standards throughout the development of our project in order to meet our desired goals. Through each step we adhered to strict documentation to ensure that each individual would be up to date with the information regarding the project. We provided our advisor with weekly updates which were additionally shared with each other. These updates were stored chronologically for ease of access for each individual and made accessing and sharing information easier. With each step of our project our goal was to ensure that each module retained its functionality with the addition of features, much like Agile development. When implementing new hardware features, each hardware pin that would be utilized was tested for the appropriate functionality first by verifying through the use of a multimeter and then implementing that pin with the desired submodule. For software development, a working version of the firmware would be stored and then the test firmware would be pushed to the drone or Raspberry Pi. The software would then be verified either through physical functional verification or through digital verification depending on the specific task being tested. If the software succeeded, it would be used as the new working firmware version. Each working version would be stored in case an unforeseen issue was noted and could be brought back to a previous, clean build of the firmware.

**Chapter 7 Testing and Experiment**

**7.1 Testing and Experiment Scope**

The test objective of this project was to ensure that we met our goals of providing achievable autonomous flight. In addition to the flight, the system would properly capture events on location. Our goal is that the system functions as designed in real time without problems or other errors. In order to achieve this, we unit tested each module to first ensure that they functioned properly individually. Once the subunit was tested, it would then be combined with other units, and then functionally tested. With this method of testing, we can prevent large errors and keep them minimized to each submodule.

This project incorporated both software and hardware units which required their own specific types of unit testing, which can be seen in the next section. To functionally verify the system, we implemented both black-box and white-box testing. For the black-box testing, we would test the functionality of the system as if we were an uninformed user on the system. For this user, they would simply select a location for the drone to fly to. For white-box testing, we test each of the test cases for the drone to go through. These test cases are similar to the states in the state transition diagram in section 3.1. We are testing each aspect of the code to make sure it adheres to the states that it is designed to transition between.

**7.2 Testing and Experiment Approach**

For this project, several tools were used in order to ensure proper function and execution of the desired application. This project takes on a combination of both hardware and software tools, and required both of these aspects to work in sync with each other. The hardware development took place before the software development; however, during the development of this project, both software and hardware development took place simultaneously.

Our initial test is to evaluate whether the CrazyFlie can handle the weight of the peripherals we want to place on the drone. After some research, it was discovered that the maximum recommended payload for the CrazyFlie is 15 grams [2]. While the Raspberry Pi Zero only comes in at a weight of 9 grams [3], this nearly exceeded the maximum payload for the remaining peripheral devices including the GPS and proximity sensors. In order to carry the payload required for this project, a second drone was acquired which could provide greater power, when given more powerful ESCs and a new LiPo battery. With these additional modifications, we generate enough power for the added weight of the Pi Zero, GPS module, proximity sensors, and the new additions of the battery and ESCs.

In order to gauge that these additional devices will integrate properly into our system, each device will need to be tested to ensure that it complies with our desired values for each respective device. To ensure that the ESCs will function properly, we specified a battery that will meet our desired voltage of 11.1V and also has enough charge capacity to sustain flight for our desired period of time of twenty-five minutes. The voltage of the battery will be tested using a multimeter to ensure we are getting a nominal voltage out of it. Additionally, to ensure that we are getting the desired values for our ESCs, we will also analyze both their voltage and current with a multimeter to ensure that we are getting an increase in power compared to the CrazyFlie alone.

On a general scale, each component will be tested with a multimeter and waveform logic analyzer to ensure the pins are soldered to specification, and are functioning as desired. These tests will be done as each component is installed. This will guarantee that no hardware problems exist in faulty pins.

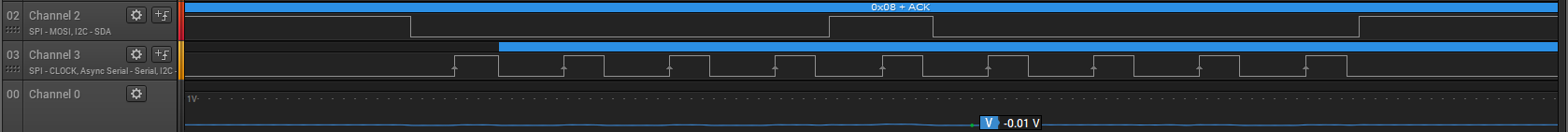
In addition to hardware testing, software testing will be conducted to ensure proper execution of the code on each device. Two of our devices require software programming, the CrazyFlie controller and the Raspberry Pi Zero. Each of these devices follow a different approach for how they are programmed because they each have a different utility.

The firmware utilized for the CrazyFlie is the PX4 autopilot software which is very popular for consumer based applications due to the fact that it is open source. Because it is open source, we have more control over what exactly the drone does and can implement more precision. In addition the PX4 firmware, we are using QGroundControl which allows us to leverage the full functionality of the open source platform of PX4 in a more user friendly setting for testing. It is an open source service so we can develop in it to utilize it uniquely for our application. It will provide us the ability to plan autonomous missions and fully customize our drone for our application.

**7.3 Testing and Experiment Results and Analysis**

We began by testing basic I/O ports for proper functionality. Each I/O port needed can be expanded into different functions: UART, I2C, and GPIO signals. For the communications between the RPi and CrazyFlie, the test criteria was: can we send messages from the RPi to the CrazyFlie, do these packets address the correct registers for the PX4 firmware to read, and does the data match the data being stored in the RPi output buffer? Separately, we needed to do this for the UART channel on the GPS module to make sure the data being transferred was accurate and correct. Lastly, we needed to ensure the signal data from the Laser Sensors were being transferred to the RPi properly according to the distance expectations. Further testing and analysis would be done on the drone once full assembly was complete. This includes ESC signal inputs from the CrazyFlie to the ESC. These signals must correspond to the autopilot controls being generated by the PX4 firmware. A proper output current must be generated from the ESC at the respective signal input value.

To ensure proper data was being written to the CrazyFlie, a logic analyzer was connected in parallel with the two separate communication busses, I2C and UART. Once we had the two channels connected, we were able to sample the data being transmitted over SDA, SCL, UART-RX, and UART-TX. The data recorded can be found in Figure 8 andFigure 9. As you can see, the waveform values in these samples, match the waveforms as specified in the manufacturer’s datasheets. These communications are vital for the drone having autonomous and object avoidance capability.



**Figure 7.** I2C communication established between RPi and CrazyFlie



**Figure 8.** UART communication established between GPS and CrazyFlie

An initial test we needed to perform prior to flying the drone was figuring out how much current was needed to drive the motors at certain RPM. The RPM to altitude relationship is not provided in this report; however, we were able to measure the motor RPM by attaching a small piece of iron to the end of the propeller tips, and reading its spin by a hall effect rotational speed sensing device. The outputs of this device were displayed on an output console, recorded and plotted here in a chart. The currents were measured using a multimeter in series with the battery circuit. The final chart for this test is shown in figure 10 which displays the relationship between the drone’s RPM readings per amperes applied. The results showed that we were in fact supplying 5A to each of the four motors. With the constant power approximately at 55W, we are able to drive the motors with enough power to lift the entire assembly into flight at rapid rates.



**Figure 9.** Chart for Motor RPM due to applied current.\*

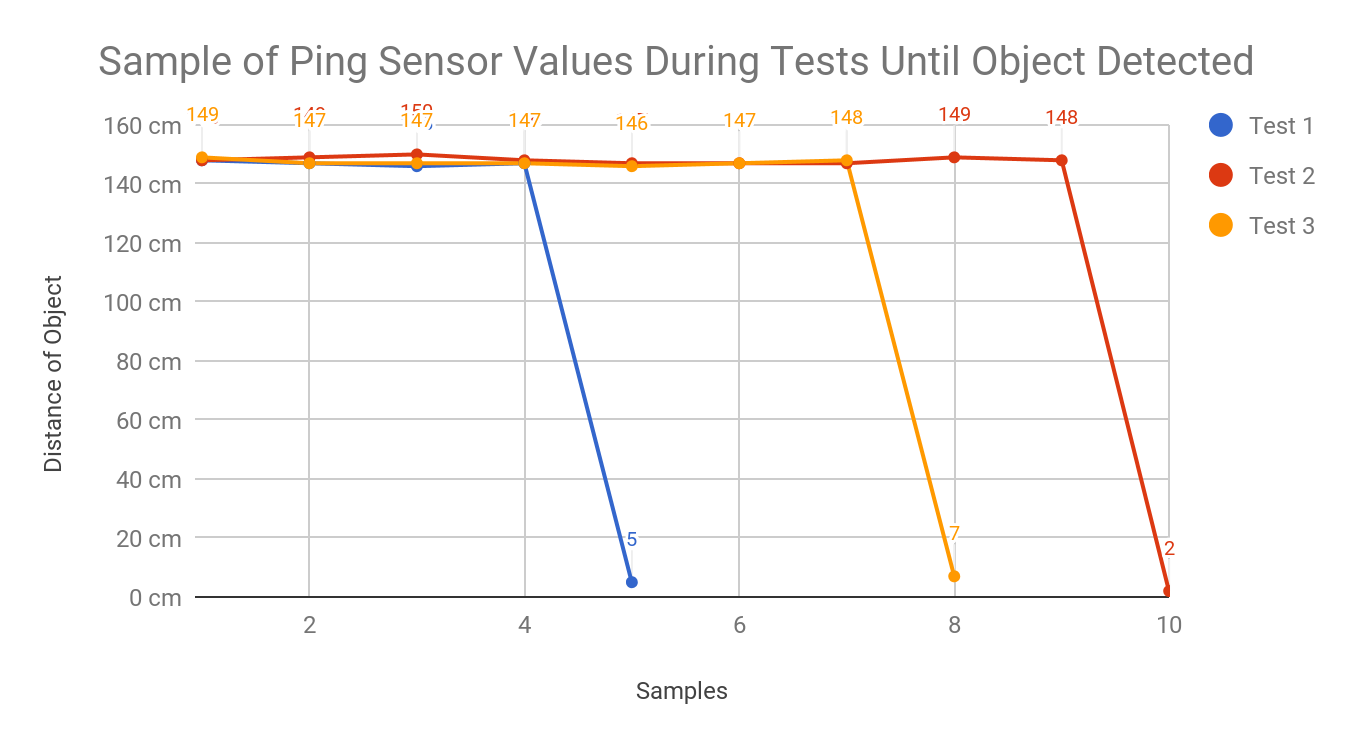
The preceding tests have to do with verifying the CrazyFlie signal outputs to the ESC. To test these signals, we used an oscilloscope to measure the signals. The duty cycle of the signals to the ESC must correlate to the power applied to the motors. As we approached 50% duty cycle, the propellers would need to be spinning at 50% of the max RPM. The drone needed to be held down to run this test to prevent damaging it during motor signal testing. What we found was that each quadrant of the drone ran at the optimal RPM per each signal wire duty cycle. This validated that our drone could run properly for elevation increase and decrease, left banks, right banks, back-up, move forward, and rotational movements.

Flight testing began with some simple manual flights that tested the endurance of the drone. We manually controlled the drone, but also set it into several different modes. The first portion of testing was just manually putting the drone under higher G forces than it should experience while performing responses. After shaking the drone we ensured the chassis and hardware components were fastened securely and could sustain heavy G forces. From here were transferred the drone into a follow me test. These tests did not include obstacle avoidance, but did include altitude scaling using the drones barometric sensor. This tested the drones ability to climb rapidly which put heavy stress on the battery. After these walk through tests were completed, we ensured the battery capacities were drained such that we could measure the endurance of the drone under heavy loads. Table 2 shows the average flight time based on the average current draw experienced through different flight conditions.

**Table 2.** Average Flight Times based on Avg Current Draw\*

| **Avg. Current** | **Battery Cap. (Ah)** | **Flights** | **Avg. Flight Time (minutes)** |
| --- | --- | --- | --- |
| 4.99 | 2.2 | 27 | 26.48 |

For the obstruction avoidance, we needed to implement the firmware program on an arduino, and hook up the ping sensor. We then powered the device on, flashed the firmware, and observed the outputs. In figure11, the firmware test program is shown. In figure 12, the ping sensor distance outputs are shown. We knew the sensors were functional and ready for integration on the RPi. The firmware was modified for RPi compatibility and tested with the drone in later configurations for obstruction avoidance.



**Figure 10.** Ping Sensor data output

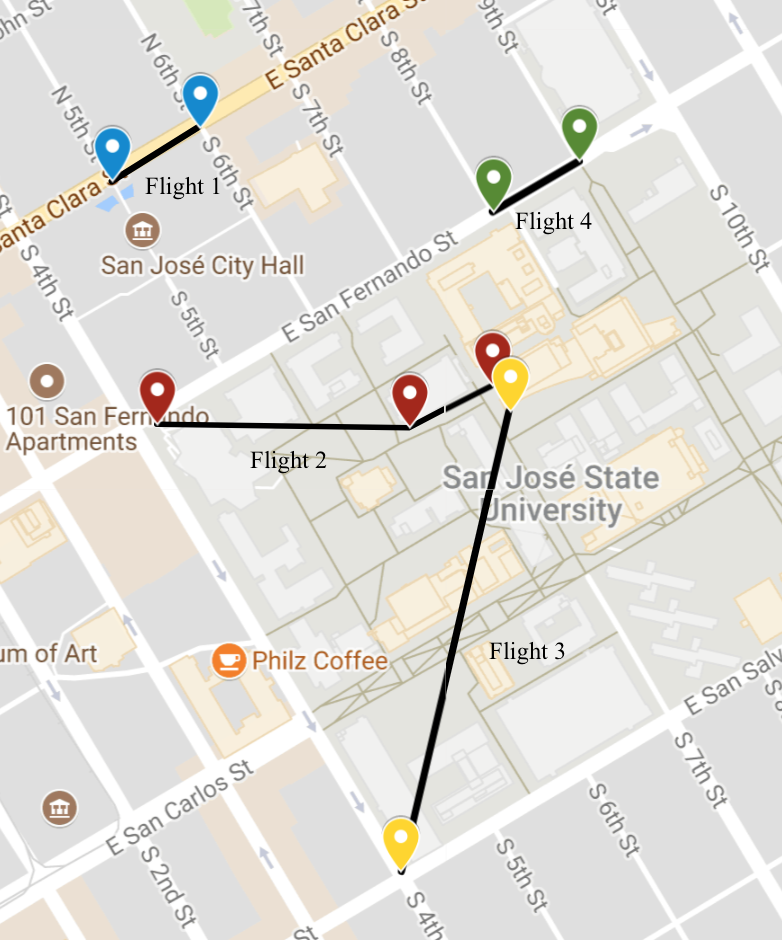
Similarly, the code was modified for testing the laser beam sensors with the Arduino. We received similar distance values at centimeter lengths. These values confirmed the operation of the laser sensors for integration with the drone. The ping sensors could be used for close proximity sensing, and the laser beam sensors could then be used for long range sensing.

The drone needed to be tested to make sure it is able to fly autonomously from one specific location to another. This is interfaced through the PX4 flight control software called QGC Control. The QGC flight control is the command station behind the drone, and it runs through a basic test loop upon startup. After the initial startup and check the drone is set to “ARMED” which means the drone is ready. The software gives the user a few options for different flight modes, we can fly the drone manually, send it on missions (autonomous flight mode) or have it ready to be calibrated. In the figure below we can see the startup check the QGC performs. It is similar to the BIOS initialization in a computer. The on-board files are checked to make sure everything is initialized properly and checks the firmware that was loaded on the memory of the microcontroller. Next it checks the radio communication with the drone. The radio that is used is the CrazyRadio PA which is based off the nRF24LU1+ from Nordic Semiconductor. It runs on the 2.4GHz frequency and can be run on both UART and SPI protocol which made it easier for us to incorporate into our project. With the default setup and parts on the CrazyFlie it comes with a compass, on-board gyroscope and accelerometer which all need to be initialized and tested before they can be used. The drivers that we are using have a built in mechanism to test the sensors before the drone is assigned the “ARMED” status. The next part of the test is to run through the hardware components of the drone to make sure the battery has enough charge to be able to perform a mission.

The uBLOX-MAX-M8Q GPS module was connected to the CrazyFlie M4-Cortex based controller. We started up the drone and were able to receive GPS coordinates of the drone at our base station. The GPS coordinates were being sent via I2C protocol, and we were able to record some of the GPS flight plan coordinates that were uploaded to the drone via the qGC. This data is shown in Table 3. These tests gave way to a critical specification that prove our drone’s ability to self-navigate, but also respond within a relatively short amount of time. When compared to the average response times of the UPD, the ARD drone outperformed them by an average of 472 seconds per emergency request - this is an average response time improvement of seven minutes and fifty two seconds.

**Table 3.** GPS Flight Information with Travel Times[[1]](#footnote-0)

| **Flight** | **° N** | **° W** | **Destination** | **Distance (mi)** | **Time (secs)** |
| --- | --- | --- | --- | --- | --- |
| 1 | 37.338 | -121.886 | From: Santa Clara & 5th St | 0.08 | 37.5 |
| 37.338 | -121.885 | To: Lee's Sandwiches |
| 2 | 37.336 | -121.885 | From: On 4th Cafe | 0.40 | 53 |
| 37.336 | -121.883 | To: Clark Hall |
| 37.336 | -121.881 | To: Student Union |
| 3 | 37.338 | -121.882 | From: San Fernando & 8th St | 0.07 | 15 |
| 37.338 | -121.881 | To: San Fernando & 9th St |
| 4 | 37.336 | -121.882 | From: Student Union | 0.43 | 62 |
| 37.332 | -121.883 | To: 4th & San Salvador |



**Figure 11.** Flight destinations of test flights

The testing that has been done thus far has proven to be a success and upon arrival of the new parts and GPS we are hoping for similar results.



**Figure 12**. Bootup Test that are run

The results of the hardware setup can be seen in the figures below. The photos show each module individually to display each the detail of each component. Many of the components were combined in the hardware chassis mounting interface.

| **Figure 12.** Hardware chassis mounting interface which houses both the Crazyflie and BigQuad expansion port with the uBlox GPS module and also houses the Raspberry Pi Zero | **Figure 13.** Underside view of the hardware chassis mounting interface mounted on the Xiro drone. The hardware chassis mounting interface is responsible for holding all of the peripherals | **Figure 14.** Drone and peripherals, which include (from left to right) uBlox GPS module, Arduino ping sensor, BitCraze BigQuad expansion board, and Raspberry Pi Zero | **Figure 15.** Internal drone core (CrazyFlie 2.0, BigQuad, and 4-channel ESC). In black is the BitCraze expansion board mounted atop the CrazyFlie controller. The black and red wires come from the ESCs which control the speed of the motors |
| --- | --- | --- | --- |

| **Figure 16.** BitCraze BigQuad Expansion Board which allows for the addition of external ESCs and allows us to mount the GPS module | **Figure 17.** UBLOX GPS Module which can be used to communicate with our CrazyFlie controller over UART, SPI, or I2C | **Figure 18.** Raspberry Pi Zero W which is used to interface the camera module as well as the Arduino ping sensor | **Figure 19.** Arduino Ping Sensor which is an ultrasonic range finder that works by sending a sound wave and finding the distance of the closest object |
| --- | --- | --- | --- |

**Chapter 8. Conclusion and Future Work**

**8.1 Conclusion**

In conclusion, our group was able to incorporate various hardware units into one programmable unit capable of responding autonomously to a distress signal. We were able to properly program each hardware device so that they could communicate seamlessly with one another to produce a working drone system. Once the system and its components were tested to be functioning properly, we were able to program the CrazyFlie controller using PX4 drone autopilot software and modify the code provided in those libraries to accurately function to our application.

This project had a slow start, however, we were able to overcome this by adding extra effort as the semester began to come to a close. Once we were able to establish an effective work time and place, progress was made much more rapidly. This project was not without its shortcomings, however. For one, we had to shift our goal from enabling a bystander the ability to activate a distress signal via their phone or another device to signal the location for the drone to travel to. Implementing a homing software would have proved too difficult for the limited time period. Instead, we opted to send the coordinates to the drone which can be interfaced with the GPS. Another difficulty was accessibility. When someone wanted to work on the drone, they would have to ask for the parts from another member and then other members wouldn’t have access to them. The most convenient option would be to work on it at school, however, this sometimes proved difficult for our group with our busy class and work schedules. This aspect definitely made this project harder compared to a software based project where work could be done remotely.

**8.2 Future Work**

This system proved that an autonomous response drone is a feasible device for the future. The applications are limitless but there may also be an increase in limitations in the future. Drones are relatively new and legislature regarding them is new as well. Currently, San Jose is a no fly zone due to the airport in San Jose. While these drones can see potential applications in the future for the purpose we intended (aiding individuals in a potential assault), they could also see you in delivery for Amazon or other services, or even delivering aid to people affected by natural disasters or soldiers.

This system that we created is just a sample of the potential of a drone with these capabilities. If this drone where to see large scale production, it would benefit greatly from combining the multiple modules in the drone. Bypassing the Raspberry Pi for the camera and thus rendering it useless would be the most logical step to provide a more seamless, cost-effective design. Also choosing the scale for which the drone needs to be would be an important factor. Depending on the needs of the drone, producing a large drone may be a more attractive option, however, if less power and flight time is required, then a lighter, smaller option would be better suited. Lastly, in order to generate greater revenue, producing a drone with proprietary parts would provide more revenue from spare parts and licensing for the software. The number of applications are endless and we may very well be seeing drones used for multiple purposes in the near future.

**References**

[1]A. Glaser, "Police departments are using drones to find and chase down suspects", *Recode*, 2017. [Online]. Available: https://www.recode.net/2017/4/6/15209290/police-fire-department-acquired-drone-us-flying-robot-law-enforcement.

[2]"projects:crazyflie2:hardware:specification [Bitcraze Wiki]", *Wiki.bitcraze.io*, 2015. [Online]. Available: https://wiki.bitcraze.io/projects:crazyflie2:hardware:specification.

[3]"Raspberry Pi Zero weight & power consumption | mitat.tuu.fi", *Mitat.tuu.fi*. [Online]. Available: http://mitat.tuu.fi/?p=423. [Accessed: 10- Aug- 2017].

[4]"doc:crazyflie:bootloader:index [Bitcraze Wiki]", *Wiki.bitcraze.io*, 2015. [Online]. Available: https://wiki.bitcraze.io/doc:crazyflie:bootloader:index.

[5]"Crazyflie 2.0", *Bitcraze Store*, 2017. [Online]. Available: https://store.bitcraze.io/products/crazyflie-2-0.

[6]G. Bekey, *Autonomous Robots: From Biological Inspiration to Implementation and Control*. Cambridge: MIT Press, 2005, p. 11.

[7]"projects:crazyflie2:architecture:index [Bitcraze Wiki]", *Wiki.bitcraze.io*, 2015. [Online]. Available: https://wiki.bitcraze.io/projects:crazyflie2:architecture:index.

[8]"STM32F405RG - High-performance foundation line, ARM Cortex-M4 core with DSP and FPU, 1 Mbyte Flash, 168 MHz CPU, ART Accelerator - STMicroelectronics", *St.com*, 2017. [Online]. Available: http://www.st.com/en/microcontrollers/stm32f405rg.html.

[9]"nRF51 Series SoC / Products / Home - Ultra Low Power Wireless Solutions from NORDIC SEMICONDUCTOR", *Nordicsemi.com*, 2017. [Online]. Available: https://www.nordicsemi.com/Products/nRF51-Series-SoC.

[10]"Mission Planner Home — Mission Planner documentation", *Ardupilot.org*, 2016. [Online]. Available: http://ardupilot.org/planner/index.html#home.

[11]"projects:crazyflie2:expansionboards:bigquad [Bitcraze Wiki]", *Wiki.bitcraze.io*, 2016. [Online]. Available: https://wiki.bitcraze.io/projects:crazyflie2:expansionboards:bigquad.

[12]L. Koh and S. Wich, "Dawn of Drone Ecology: Low-Cost Autonomous Aerial Vehicles for Conservation", *Tropical Conservation Science*, vol. 5, no. 2, pp. 121-132, 2012.

[13]Z. Yu, K. Nonami, J. Shin and D. Celestino, "3D Vision Based Landing Control of a Small Scale Autonomous Helicopter", *International Journal of Advanced Robotic Systems*, vol. 4, no. 1, p. 7, 2007.

[14]R. Faragher, O. Chick and D. Wagner, "Captain Buzz: An all-smartphone autonomous delta-wing drone", *Workshop on Micro Aerial Vehicle Networks, Systems, and Applications (with ACM MOBISYS)*, 2015.

[15]Craig, Tim; Berman, Mark; Barrett, Devlin; Zapotosky, Matt (October 3, 2017). "Police say Las Vegas gunman planned 'extensively,' used cameras to monitor police as they approached". The Washington Post. Retrieved October 4, 2017.

[16]Glaser, A. (2016, July 24). 11 Police Robots Patrolling Around the World. Retrieved November 30, 2017, from <https://www.wired.com/2016/07/11-police-robots-patrolling-around-world/>

[17]M. Saska, T. Krajník, J. Faigl, V. Vonásek and L. Přeučil, "Low cost MAV platform AR-drone in experimental verifications of methods for vision based autonomous navigation," 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, 2012, pp. 4808-4809.

doi: 10.1109/IROS.2012.6386277

[18]Lugo, J. J., & Zell, A. (2014). Framework for autonomous on-board navigation with the AR. Drone. Journal of Intelligent & Robotic Systems, 73(1-4), 401-412.

[19]Tang, Lina & Shao, Guofan. (2015). Drone remote sensing for forestry research and practices. Journal of Forestry Research. 26. 791-797. 10.1007/s11676-015-0088-y.

[20]Autopilot, P. P. (n.d.). Full Parameter Reference. Retrieved October 06, 2017, from https://docs.px4.io/en/advanced\_config/parameter\_reference.html

**Appendices**

**Appendix - A:** *Source Code*

**.gitmodules**

[submodule "Bootloader"]  
 path = Bootloader  
 url = https://github.com/PX4/Bootloader  
[submodule "Firmware"]  
 path = Firmware  
 url = https://github.com/PX4/Firmware

**cam.py**

from picamera import PiCamera  
from time import sleep  
import argparse  
import datetime as dt  
import os  
import sys  
  
camera = PiCamera()  
camera.vflip = True  
camera.brightness = 70  
camera.sharpness = 40  
camera.contrast = 60  
camera.saturation = 60  
camera.exposure\_mode = 'auto'  
  
  
def record(rec\_time, file\_path):  
 camera.start\_recording(file\_path)  
 sleep(rec\_time)  
 camera.stop\_recording()  
  
  
def live\_feed(feed):  
 # live feed  
 while feed:  
 camera.start\_preview()  
 sleep(10)  
 camera.stop\_preview()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 input\_parser = argparse.ArgumentParser()  
 input\_parser.add\_argument("time", type=int,  
 help="Recording Time")  
 input\_parser.add\_argument("-l", "--live", action="store\_true",  
 help="live Feed")  
 args = input\_parser.parse\_args()  
 if args.live:  
 live\_feed(args.live)  
 else:  
 print("Recording")  
 file\_path = str(dt.date) + 'video.h264'  
 record(args.time, file\_path)

**req.txt**

empy==3.3.2  
Jinja2==2.9.6  
MarkupSafe==1.0  
numpy==1.13.1  
pandas==0.20.3  
pyserial==3.4  
python-dateutil==2.6.1  
pytz==2017.2  
six==1.10.0

**mac\_setup**

#!/bin/bash  
  
sudo -v #Ask for the administrator password upfront  
  
###############################################################################  
# ========= Xcode ========== #  
###############################################################################  
# xcode-select --install #Install Command Line Tools  
# sudo xcodebuild -license #Agree to the XCode license  
  
if ! xcode-select --print-path &> /dev/null; then  
 xcode-select --install &> /dev/null #install cli tools if needed  
  
 until xcode-select --print-path &> /dev/null; do  
 sleep 5  
 done  
  
 print\_result $? 'Install XCode Command Line Tools'  
  
 sudo xcode-select -switch /Applications/Xcode.app/Contents/Developer  
 print\_result $? 'Make "xcode-select" developer directory point to Xcode'  
  
 sudo xcodebuild -license #Accept User Agreement  
 print\_result $? 'Agree with the XCode Command Line Tools licence'  
fi  
  
###############################################################################  
# ========= Brew ========== #  
###############################################################################  
# Check for Homebrew and install it if missing + list of dev packages  
if test ! $(which brew)  
then  
 echo "Installing Homebrew..."  
 ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"  
fi  
  
brew update # Latest updates and upgrades  
brew upgrade # Latest updates and upgrades  
brew cleanup # Remove Depracted formulas  
brew prune # Remove Depracted formulas  
  
echo " ───────────────────────────────────────────────────┐"  
echo " Installing 'BREW Packages' "  
echo "└─────────────────────────────────────────────────── "  
brew tap px4/px4  
brew install px4-dev  
brew install python2  
  
echo " ───────────────────────────────────────────────────┐"  
echo " Installing Python packages "  
echo "└─────────────────────────────────────────────────── "  
pip2 install --upgrade pip  
pip2 install --upgrade virtualenv  
pip2 install -r requirements.txt

**.gitignore**

# Covers JetBrains IDEs: IntelliJ, RubyMine, PhpStorm, AppCode, PyCharm, CLion, Android Studio and Webstorm  
  
# User-specific stuff:  
.idea/\*\*/tasks.xml  
.idea/dictionaries  
  
# Sensitive or high-churn files:  
.idea/\*\*/dataSources/  
.idea/\*\*/dataSources.ids  
.idea/\*\*/dataSources.xml  
.idea/\*\*/dataSources.local.xml  
.idea/\*\*/sqlDataSources.xml  
.idea/\*\*/dynamic.xml  
.idea/\*\*/uiDesigner.xml  
  
# Gradle:  
.idea/\*\*/gradle.xml  
.idea/\*\*/libraries  
  
# CMake  
cmake-build-debug/  
  
# Mongo Explorer plugin:  
.idea/\*\*/mongoSettings.xml  
  
## File-based project format:  
\*.iws  
  
## Plugin-specific files:  
  
# IntelliJ  
out/  
  
# mpeltonen/sbt-idea plugin  
.idea\_modules/  
  
# JIRA plugin  
atlassian-ide-plugin.xml  
  
# Cursive Clojure plugin  
.idea/replstate.xml  
  
# Crashlytics plugin (for Android Studio and IntelliJ)  
com\_crashlytics\_export\_strings.xml  
crashlytics.properties  
crashlytics-build.properties  
fabric.properties  
  
# Python specific .gitignore  
  
# Byte-compiled / optimized / DLL files  
\_\_pycache\_\_/  
\*.py[cod]  
  
# C extensions  
\*.so  
  
# Distribution / packaging  
.Python  
env/  
bin/  
build/  
develop-eggs/  
dist/  
eggs/  
lib/  
lib64/  
parts/  
sdist/  
var/  
\*.egg-info/  
.installed.cfg  
\*.egg  
  
# Installer logs  
pip-log.txt  
pip-delete-this-directory.txt  
  
# Unit test / coverage reports  
htmlcov/  
.tox/  
.coverage  
.cache  
nosetests.xml  
coverage.xml  
  
# Translations  
\*.mo  
  
# Mr Developer  
.mr.developer.cfg  
.project  
.pydevproject  
  
# Rope  
.ropeproject  
  
# Django stuff:  
\*.log  
\*.pot  
  
# Sphinx documentation  
docs/\_build/  
  
#Custom  
.idea/\*  
venv/

**gps.c**

/\*  
 \* || \_\_\_\_ \_ \_\_  
 \* +------+ / \_\_ )(\_) /\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_ \_\_\_  
 \* | 0xBC | / \_\_ / / \_\_/ \_\_\_/ \_\_\_/ \_\_ `/\_ / / \_ \  
 \* +------+ / /\_/ / / /\_/ /\_\_/ / / /\_/ / / /\_/ \_\_/  
 \* || || /\_\_\_\_\_/\_/\\_\_/\\_\_\_/\_/ \\_\_,\_/ /\_\_\_/\\_\_\_/  
 \*  
 \* Crazyflie control firmware  
 \*  
 \* Copyright (C) 2011-2012 Bitcraze AB  
 \*  
 \* This program is free software: you can redistribute it and/or modify  
 \* it under the terms of the GNU General Public License as published by  
 \* the Free Software Foundation, in version 3.  
 \*  
 \* This program is distributed in the hope that it will be useful,  
 \* but WITHOUT ANY WARRANTY; without even the implied warranty of  
 \* MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
 \* GNU General Public License for more details.  
 \*  
 \* You should have received a copy of the GNU General Public License  
 \* along with this program. If not, see <http://www.gnu.org/licenses/>.  
 \*  
 \* exptest.c - Testing of expansion port.  
 \*/  
#define DEBUG\_MODULE "GTGPS"  
  
#include <stdint.h>  
#include <string.h>  
#include <stdio.h>  
  
#include "stm32fxxx.h"  
#include "config.h"  
#include "console.h"  
#include "uart1.h"  
#include "debug.h"  
#include "deck.h"  
#include "FreeRTOS.h"  
#include "task.h"  
#include "queue.h"  
#include <string.h>  
#include <stdlib.h>  
#include <math.h>  
#include "log.h"  
  
static bool isInit;  
  
#define LEN\_TOKEN 5  
#define MAX\_LEN\_SENTANCE 100  
char buff[MAX\_LEN\_SENTANCE];  
uint8\_t bi;  
  
typedef bool (\*SentanceParser)(char \* buff);  
  
typedef struct {  
 const char \* token;  
 SentanceParser parser;  
} ParserConfig;  
  
typedef enum {  
 FixNone = 1,  
 Fix2D = 2,  
 Fix3D = 3  
} FixQuality;  
  
typedef enum {  
 NoFix = 1,  
 GPSFix = 2  
} FixType;  
  
typedef enum {FIELD\_COORD, FIELD\_FLOAT, FIELD\_INT} FieldType;  
  
typedef struct {  
 FixQuality fix;  
 uint32\_t locks[12];  
 float pdop;  
 float hdop;  
 float vdop;  
} Basic;  
  
typedef struct {  
 uint32\_t fixtime;  
 int32\_t latitude;  
 int32\_t longitude;  
 FixType fixtype;  
 uint32\_t nsat;  
 float hdop;  
 float alt;  
 float height;  
} MeasData;  
  
static Basic b;  
static MeasData m;  
  
// Only use on 0-terminated strings!  
static int skip\_to\_next(char \*\* sp, const char ch) {  
 int steps;  
 while (ch != 0 && (\*\*sp) != ch) {  
 (\*sp)++;  
 steps++;  
 }  
 if (ch != 0)  
 (\*sp)++;  
 return (ch != 0 ? steps : -1);  
}  
  
static int32\_t parse\_coordinate(char \*\* sp) {  
 int32\_t dm;  
 int32\_t degree;  
 int32\_t minute;  
 int32\_t second;  
 int32\_t ret;  
 char \* i;  
 char \* j;  
  
 // Format as DDDMM.SSSS converted by long or lat = DDD + MM / 100 + SSSS/3600  
 // To avoid inaccuracy caused by float representation save this value as  
 // a large number \* 10 M  
  
 // 32 18.0489 N = 32 degrees + 18.0489 / 60 = 32.300815 N  
 dm = strtol(\*sp, &i, 10);  
 degree = (dm / 100) \* 10000000;  
 minute = ((dm % 100) \* 10000000) / 60;  
 second = (strtol(i+1, &j, 10) \* 1000) / 60;  
 ret = degree + minute + second;  
 skip\_to\_next(sp, ',');  
 if (\*\*sp == 'S' || \*\*sp == 'W')  
 ret \*= -1;  
 return ret;  
}  
  
static float parse\_float(char \* sp) {  
 float ret = 0;  
 int major = 0;  
 int minor = 0;  
 int deci\_nbr = 0;  
 char \* i;  
 char \* j;  
  
 major = strtol(sp, &i, 10);  
 // Do decimals  
 if (strncmp(i, ".", 1) == 0) {  
 minor = strtol(i+1, &j, 10);  
 deci\_nbr = j - i - 1;  
 }  
 ret = (major \* pow(10, deci\_nbr) + minor) / pow(10, deci\_nbr);  
 //printf("%i.%i == %f (%i) (%c)\n", major, minor, ret, deci\_nbr, (int) \*i);  
 return ret;  
}  
  
static void parse\_next(char \*\* sp, FieldType t, void \* value) {  
 skip\_to\_next(sp, ',');  
 //DEBUG\_PRINT("[%s]\n", (\*sp));  
 switch (t) {  
 case FIELD\_INT:  
 \*((uint32\_t\*) value) = strtol(\*sp, 0, 10);  
 break;  
 case FIELD\_FLOAT:  
 \*((float\*) value) = parse\_float(\*sp);  
 break;  
 case FIELD\_COORD:  
 \*((int32\_t\*) value) = parse\_coordinate(sp);  
 }  
}  
  
static bool gpgsaParser(char \* buff) {  
 int i = 0;  
 char \* sp = buff;  
  
 // Skip leading A/M  
 skip\_to\_next(&sp, ',');  
  
 parse\_next(&sp, FIELD\_INT, &b.fix);  
 for (i = 0; i < 12; i++) {  
 parse\_next(&sp, FIELD\_INT, &b.locks[i]);  
 }  
 parse\_next(&sp, FIELD\_FLOAT, &b.pdop);  
 parse\_next(&sp, FIELD\_FLOAT, &b.hdop);  
 parse\_next(&sp, FIELD\_FLOAT, &b.vdop);  
  
 //dbg\_print\_basic(&b);  
 return false;  
}  
  
static bool gpggaParser(char \* buff) {  
 char \* sp = buff;  
  
 parse\_next(&sp, FIELD\_INT, &m.fixtime);  
 parse\_next(&sp, FIELD\_COORD, &m.latitude);  
 parse\_next(&sp, FIELD\_COORD, &m.longitude);  
 parse\_next(&sp, FIELD\_INT, &m.fixtype);  
 parse\_next(&sp, FIELD\_INT, &m.nsat);  
 parse\_next(&sp, FIELD\_FLOAT, &m.hdop);  
 parse\_next(&sp, FIELD\_FLOAT, &m.alt);  
 skip\_to\_next(&sp, ',');  
 // Unit for altitude (not used yet)  
 parse\_next(&sp, FIELD\_FLOAT, &m.height);  
 skip\_to\_next(&sp, ',');  
 // Unit for height (not used yet)  
 skip\_to\_next(&sp, ',');  
 //consolePutchar('.');  
 //consoleFlush();  
 return false;  
}  
  
static ParserConfig parsers[] = {  
 {.token = "GPGSA", .parser = gpgsaParser},  
 {.token = "GPGGA", .parser = gpggaParser}  
};  
  
static bool verifyChecksum(const char \* buff) {  
 uint8\_t test\_chksum = 0;  
 uint32\_t ref\_chksum = 0;  
 uint8\_t i = 0;  
 while (buff[i] != '\*' && i < MAX\_LEN\_SENTANCE-3) {  
 test\_chksum ^= buff[i++];  
 }  
 ref\_chksum = strtol(&buff[i+1], 0, 16);  
  
 return (test\_chksum == ref\_chksum);  
}  
  
static uint8\_t baudcmd[] = "$PMTK251,115200\*1F\r\n";  
  
// 5 Hz  
static uint8\_t updaterate[] = "$PMTK220,200\*2C\r\n";  
static uint8\_t updaterate2[] = "$PMTK300,200,0,0,0,0\*2F\r\n";  
  
// 10 Hz  
//static uint8\_t updaterate3[] = "$PMTK220,100\*2F\r\n";  
//static uint8\_t updaterate4[] = "$PMTK300,100,0,0,0,0\*2C\r\n";  
  
  
void gtgpsTask(void \*param)  
{  
 char ch;  
 int j;  
  
 uart1SendData(sizeof(baudcmd), baudcmd);  
  
 vTaskDelay(500);  
 uart1Init(115200);  
 vTaskDelay(500);  
  
 uart1SendData(sizeof(updaterate), updaterate);  
 uart1SendData(sizeof(updaterate2), updaterate2);  
  
// uart1SendData(sizeof(updaterate3), updaterate3);  
// uart1SendData(sizeof(updaterate4), updaterate4);  
  
  
 while(1)  
 {  
 uart1Getchar(&ch);  
 consolePutchar(ch);  
  
 if (ch == '$') {  
 bi = 0;  
 } else if (ch == '\n') {  
 buff[bi] = 0; // Terminate with null  
 if (verifyChecksum(buff)) {  
 //DEBUG\_PRINT("O");  
 for (j = 0; j < sizeof(parsers)/sizeof(parsers[0]); j++) {  
 if (strncmp(parsers[j].token, buff, LEN\_TOKEN) == 0) {  
 parsers[j].parser(&buff[LEN\_TOKEN]);  
 }  
 }  
 }  
 } else if (bi < MAX\_LEN\_SENTANCE) {  
 buff[bi++] = ch;  
 }  
 }  
}  
  
  
static void gtgpsInit(DeckInfo \*info)  
{  
 if(isInit)  
 return;  
  
 DEBUG\_PRINT("Enabling reading from GlobalTop GPS\n");  
 uart1Init(9600);  
  
 xTaskCreate(gtgpsTask, "GTGPS",  
 configMINIMAL\_STACK\_SIZE, NULL, /\*priority\*/1, NULL);  
  
 isInit = true;  
}  
  
static bool gtgpsTest()  
{  
 bool status = true;  
  
 if(!isInit)  
 return false;  
  
 return status;  
}  
  
static const DeckDriver gtgps\_deck = {  
 .vid = 0xBC,  
 .pid = 0x07,  
 .name = "bcGTGPS",  
  
 .usedPeriph = 0,  
 .usedGpio = 0, // FIXME: Edit the used GPIOs  
  
 .init = gtgpsInit,  
 .test = gtgpsTest,  
};  
  
DECK\_DRIVER(gtgps\_deck);  
  
LOG\_GROUP\_START(gps)  
LOG\_ADD(LOG\_INT32, lat, &m.latitude)  
LOG\_ADD(LOG\_INT32, lon, &m.longitude)  
LOG\_ADD(LOG\_FLOAT, hMSL, &m.height)  
LOG\_ADD(LOG\_FLOAT, hAcc, &b.pdop)  
LOG\_ADD(LOG\_INT32, nsat, &m.nsat)  
LOG\_ADD(LOG\_INT32, fix, &b.fix)  
LOG\_GROUP\_STOP(gps)

1. Data used is standardized and nominal to drone specifications [↑](#footnote-ref-0)