Fire Extinguishing Unmanned Aerial Vehicle (FXUAV)

Adam Kutchak, Luis Brum, Jamie Peck, Greg Kelso

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — The emergence of drones and UAVs in recent years has opened up the possibilities of new and interesting engineering solutions to problems in the world. The FXUAV is a project that seeks apply this technology to firefighting. It uses a Logitech webcam, some clever programming, and a payload release design, to detect and extinguish flames. The FXUAV consists of three major subsystems: 1) a quadcopter used to search through the air 2) a processor that uses image recognition software to identify a fire 3) a payload release system that extinguishes the fire.**

***Index Terms* — Image processing, microcontrollers, object detection, satellite navigation systems, unmanned aerial vehicles, wireless communication.**

I. Introduction

The introduction of UAVs has expanded the range of problems that engineers are able to solve, in addition to revisiting solutions to old problems. Firefighting has always been a dangerous profession, and the creation of vehicles that are able to fly autonomously introduces a new way of thinking in regards to designing a safer fire extinguishing system. By attaching a camera to a drone, it is possible to safely conduct reconnaissance and identify the scope and severity of a fire. With increases in battery life and thrust as technology improves, drones will also be able to carry substantial payloads in the future.

The FXUAV is a step toward this safer firefighting technology. It utilizes three main subsystems consisting of a quadcopter, a camera and onboard computer for image processing, and a payload release system. These subsystems combine to solve the task of deploying a drone in the vicinity of a fire, flying until a fire is seen by the camera, and then extinguishing the fire. Throughout the entire process, a live video stream of the mission from the camera will be viewable on a user's computer. The details of these subsystems shall be covered in the sections below.

II. System Overview

The Fire Extinguisher Unmanned Aerial Vehicle will be composed of four main systems, the Quadcopter, a ground control system, the Image Detection System, and the Fire Extinguisher. The three systems will need to work simultaneously so the FXUAV can detect and extinguish the fire successfully.

A. Quadcopter

Even though there are many pre-built quadcopter systems, the FXUAV is built using parts that would best fit the objective of the FXUAV project. The Quadcopter is composed of the following items: 1) Tarot Ironman 650 Carbon Fiber Frame 2) Four Tiger-RC MN3515 KV400 Motors 3) Four 15x5 Carbon Fiber Propellers 4) Four ARRIS 3-6S 30A Electronic Speed Controllers 5) Pixhawk Flight Controller Unit 6) 3DR SIK Telemetry Radio system 7) 3DR uBlox GPS 8) Radiolink R10D radio controller Receiver 9) Power Distribution Board 10) QuadPro Power 10,000mAH 6S LiPo Battery. These different items have been flight tested to verify that the quadcopter could achieve the goal of the FXUAV project. With these different items the quadcopter was able to lift over the 10 pounds of weight necessary, and was able to fly over 10 minutes in a 200 square feet area while maintaining in contact with the Ground Control Unit. The Ground Control Unit is what controls the initial and different flight modes of the quadcopter. These are the components that the Ground Control Unit is made of these items: 1) Laptop with Mission Planner software 2) 3DR SIK Telemetry Radio system 3) Radiolink AT10 radio controller Transmitter. The Mission planner software version 1.3.38 which is made by Michael Oborne is the system that the PixHawk is programmed from, which communicates with the Pixhawk through the 3DR Sik Radio telemetry system. Prior to the initial flight the Pixhawk needs to be programmed with different flight modes, which are triggered by the AT10 radio control transmitter and allows the quadcopter to be controlled by the Fire Detection system.

B. Image Detection System

The Image Control System will be the main operator after the quadcopter is in the air. It will be responsible for detecting the fire and guiding the quadcopter to the location of the fire. See Figure 1 below for reference of how the system connectivity will be implemented.

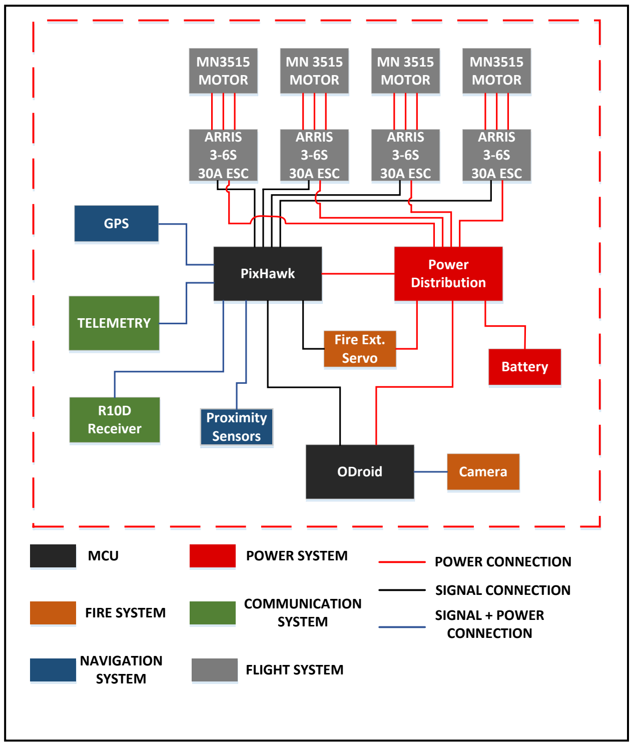


Figure 1

The Image Detection System will be composed of these different items: 1) Odroid XU4 Microcontroller Unit 2) Logitech C270 Web Camera. The Logitech web camera will be connected to the to the Odroid XU4 microcontroller unit, that will capture live video of the area which the quadcopter is flying in. The Odroid will then use a classifier to detect the fire and using pixel data it will use an algorithm that will guide the quadcopter to the fire using Mavproxy command. Once it has determined that the quadcopter is on top of the fire, it will send a command to activate the Fire Extinguisher System.

C. *Fire Extinguishing System*

The Fire Extinguishing System will release a chemical fire suppressant that can put out a fire that is at least of one square inch in size. The Fire Extinguishing will be composed of these items: 1) Chemical fire suppressant powder 2) 3D printed container system 3) 5V servo. Once the quadcopter is on top of the fire, the Odroid will send a signal to the Pixhawk which will then send a signal to the servo. Once the servo receives the signal it will allow the fire suppressant powder to disperse on top of the fire to extinguish it.

III. Quadcopter

A. *Hardware*

The quadcopter is a 4 rotor aerial vehicle that will be controlled by a Pixhawk flight controller. It supports UART, I2C, SPI, and CAN for additional peripherals, with which we will use to communicate sensor data. It also features 14 pwm/servo outputs with which we will assign four of our motors to, making it a quadcopter. Some of the embedded sensors include a gyroscope to assist stabilization, an accelerometer, and a barometer. The UART (serial) ports will be used for connections such as power. A PPM encoder will be used in order to receive our radio signal and communicate with Pixhawk's sbus input and output. The Pixhawk consumes up to 5.4V on the power module for our application and is rated for a maximum of 0V – 20V undamaged.

Connected to the Pixhawk are several peripheral components, such as a u-Blox GPS, 3DR Telemetry radio, PPM encoder, and a Radiolink AT10 receiver. The external GPS unit is a very accurate unit that operates at 3.3V. The telemetry radio allows for 2 way communication between the ground control station and the quadcopter. Messages are sent using a MAVLink protocol which is a header only language used to communicate with the Pixhawk. Lastly, the AT10 receiver and ppm encoder are used to receive input from the Radiolink AT10 transmitter. The transmitter sends pulse-width modulation signals that the Pixhawk protocol isn't compatible with. See Figure 2 below for an overview of how the hardware is assembled.

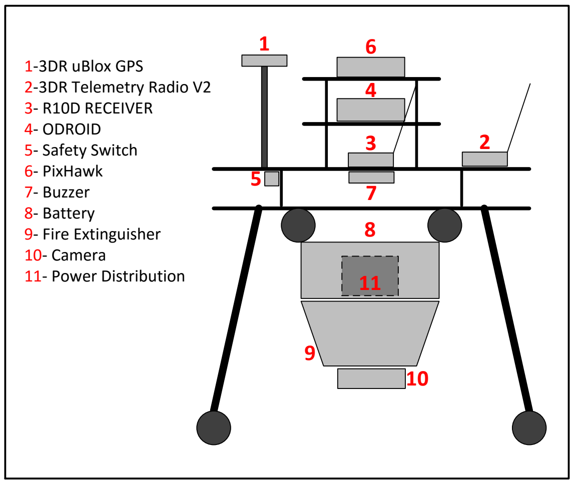


Figure 2

The PPM encoder combines those PWM signals into a single signal of with the Pixhawk can understand. These inputs are used to change flight modes, or flight path.

The quadcopter will be powered by a 22.2V, 10Ah lithium polymer battery connected to a power distribution board that is created specifically for our application and designed by us. The power distribution is responsible for distributing the supplied power to all of our components on the quadcopter.

The electronic speed controllers are used to control the rotations per minute of each motor independently, and are calibrated through Mission Planner software using Arducopter firmware. At full throttle the ESCs will be supplying the motors with 20A which translates to 2.8kg of thrust per motor. At this point we're operating at the least efficient of 6.73 grams per watt.

B. Software

Communication between the Pixhawk and all other modules will be using the MAVlink protocol, which is an acronym for Micro Air Vehicle Link. MAVLink is a header only library that uses a system ID, a component ID, a message ID, and parameters. The system ID represents the vehicle being communicated with; the component ID corresponds to the components with which information is being extracted; the message ID and parameters include what piece of info is needed. These messages are what will command the quadcopter during its mission.

MAVproxy is the application we will use to start our missions. MAVproxy is a command line based application written in python that will serve as our mission control. Because it is lightweight and simple, we will use it to send commands to our quadcopter during a mission.

IV. IMAGE DETECTION SYSTEM

A. Hardware

1. Odroid XU4

All image processing for the FXUAV System will be done via an onboard microcontroller.

2. Logitech C270

A camera will act as the eyes of the FXUAV during both the system's detection and tracking mode. For this reason, camera choice is a vital aspect of the design and development process for the FXUAU. The camera will be mounted to front side of the quadcopter frame and situated in such a way to direct its field of view straight at a strict 90-degree angle with the ground, ensuring no angular offset. The chosen camera must be able to communicate with an onboard microcontroller unit in order to transmit individual frames of data or images. The camera must also be able to provide adequate data, both in terms of rate and level of quality, so that a fire can successfully be distinguished from its surroundings. Furthermore, the quality of the cameras output is also important because it must be high enough to allow the object to be detected from a height of about 10ft. optionally, however preferred, the camera must incorporate a stabilization module to compensate for both climate 25 conditions such a wind, and vibrations stemming from the body of the quadcopter of which its attached to.

The Logitech C270 camera will be used to capture the real time video of the drone's path. This camera provides several capabilities that justify its use for the image detection and tracking components of the FXUAV. This camera proved HD and 1280 x 720 pixel video capture. It provides software enhanced photos of up to 3.0 megapixels and utilizes Logitech's Fluid Crystal Technology.

Logitech Fluid Crystal Technology encompasses a wide range of features that our image processing subsystem requires for low false detection ratings. The Logitech C270 follows the H.264 AVC Compression standard which allows for considerably lower data bit rates for high quality video capture. High definition video requires one gigabit of data per second to be transferred however the Logitech C270 can provide the same high definition video at a much smaller bit rate. The Logitech Fluid Crystal Technology also incorporates auto focus features and enhanced optics.

The Logitech C270 has a 2.4 GHz Intel Core2 Duo processor and requires 2GB of RAM as well as 200MB of hard drive space. It is USB 2.0 certified and has a video upload speed of 1Mbps. The output screen resolution is 1280 x 720 pixels and it is responsive to multiple operating systems such as Windows and Linux platforms. Lastly, this camera has an automatic light correction feature that allows it to intelligently adjust, according the current environment light settings.

B. Software

The FXUAV will contain an image processing module which serves the purpose of analyzing data to look for a specified object. The object, in the case of the FXUAV, is a small grease fire. The image processing module will receive data, or frames of images, from the Logitech C270 camera via the Odroid XU4 micro processing unit; the Odroid controller will process these frames onboard.

Once the Logitech C270 sends the images to the Odroid XU4 and, OpenCV will be used to analyze each image and detect if the fire is present in the field of view or not. If the fire is not detected within the current image, it will continue extracting frames from the video stream, otherwise, the Odroid XU4 would send a MAVLink command to the flight controller. The image processing system would then continue extracting frames from the stream and as long as the fire is detected in the images, then it will continue to report back to the flight controller accordingly via MAVLink commands.

The Intel Open Computer Vision Library, OpenCV, offers an assortment of algorithms related to both computer vision and video capturing. The libraries offered, provide a means to streamline the process of extracting and reading images from a specified video camera. OpenCV is compatible with a specific list of microcontroller units, the Odroid being one that list. Once the compliant MCU has access to the specified data source, OpenCV can execute its algorithms on that data. OpenCV's open source framework supports a select few programming language interfaces; Python, C, C++, and Java. In regards to image processing algorithms specifically, OpenCV offers two different methods for object detection; Latent SVM and Cascade Classifier. Latent SVM can only be implemented via C and C++ code. This algorithm approach is based very similarly on the Dalal-Triggs detector. It provides the programmer with convenient features such as storing images, and a framework for using filters on histogram of oriented gradients (HOG) descriptors. The second image processing algorithm provided by OpenCV, Cascade Classifier, is a two stage process. It involved the training stage and the detection stage. Amongst the two applications within OpenCV, which are used for classifier training, both HAAR-like features and LBP features are supported. Both negative and positive samples are needed when using OpenCV's Cascade Classifier Training method for object detection. Cascade Classifier Training can be implemented in all supported interfaces of OpenCV; that is, Python, Java, C, or C++. That being said, the FXUAV image processing unit will implement OpenCV's Cascade Classifier Training algorithms to allow for more versatility when it comes 31 to implementation methods. A list of classifiers will be trained to detect an object specified by a specific set of training materials. The classifiers are then used to determine, when given an image, if the specified object occurs in any vicinity within that image. An object is said to be detected, if or when a vicinity, within the image of which was passed to the classifier(s), passes all classifiers successfully. Two common errors that are seen when Cascade Classifier Training is implemented are, false positives and false negatives. A false positive error occurs when a vicinity with an image passes each classifier in indication that the object was detected when in fact, the object does not occur within that vicinity. Secondly, a false negative occurs when a vicinity within an image that does contain the specified object, does not pass all the classifiers, which again, indicates that the object does not exist in that vicinity. In a perfect world, one-hundred percent of the true positives should be deemed as true by each of the classifiers. In retrospect however, it is a fact that not every true positive can be found and each classifier is therefore trained to catch as many true positives as possible. That being said, the main issue with this method of classifier training, lies principally in the fact that the number of false positives reported by a single classifier is, on average, about fifty percent. In attempt to surmount such a high error rate, it is of practice to cascade a number of classifiers amongst the input data. Previously as mentioned, a classifier in itself has roughly a 50% false positive error rate. Therefore, by cascading several classifiers, each with an approximate 50% error rate, the overall error rate will be less; depletion occurring as the number of cascaded classifiers increases. In effort to do so, each subsequent classifier, in the cascaded line, should be trained to anticipate false positive reports from the preceding classifier and mark it as such. The AdaBoost algorithm provides the foundation of which such classifier training is a derivative of. The technique implicates each classifier based on the number of image regions reported as positive, in descending order; more explicitly, each classifier is cascaded in a decreasing manner, with respect to the number of positive image vicinities it identifies. This systematic technique allows for the number of false positives to decrease as the data gets forwarded further down the line of cascaded classifiers in a liquidating fashion. With redistribution rights for the following figure still pending from Jeff Bier, the founder of Embedded Vision Alliance, Figure 3 illustrates the cascade of classifiers of which the systematic technique executes.

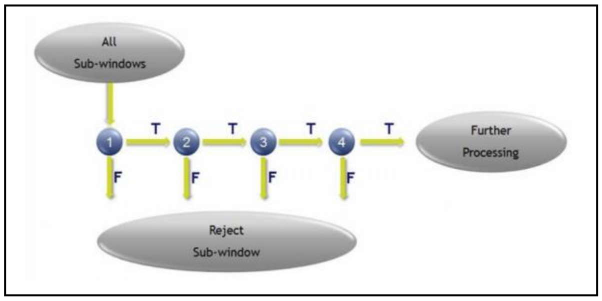


Figure 3

Each classifier is said to take in a certain number and type of feature based on the algorithm used in implementation. The basic classifier input consists of edge features, line features, and center-surround features. This set of classifier inputs are known as Haar-like features and can be seen in Figure 4 below (redistribution rights granted from OpenCV User Guide documentation).

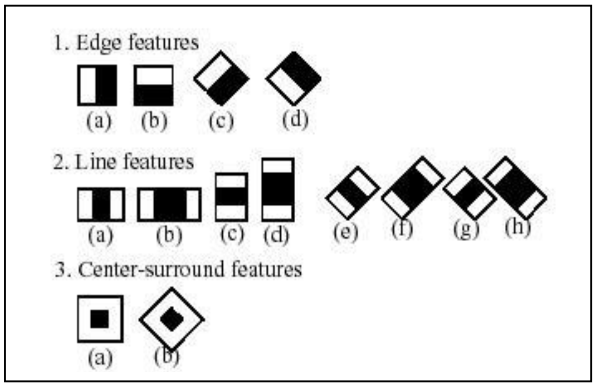


Figure 4

The white areas within each feature signify positive pixel weights and similarly, the black areas signify negative pixel weights. Responses are calculated via a simple arithmetic difference equation. The weighted sum of pixels covering the black spaces of the feature is subtracted from the weighted sum of pixels covering the whole feature. The resulting integer is then used to classify the image as either a positive or a negative, as it pertains to containing the object or not containing the object, respectively. The threshold defining a positive versus a negative response is determined by the programmer in accordance with the preferred false positive rate as well as the desire detection rate.

The pixel values that are used in the response classification process are calculated via integral images at a very rapid speed. The intrinsic functions within the OpenCV library that calculate the integral of an image, take in certain parameters from a particular set. The differing particulars are dependent upon the programming language interface being used to execute the integral function. This particular set includes and image parameter, a sum parameter, a sqsum parameter, a tilted parameter, and finally an sdepth parameter. The integral function is responsible for calculating three differing integral images for a given source image using a series of three equations (see Figure 5: Equations for Integral Image Calculation—copyright permission granted from OpenCV User Guide documentation).

$$\mathtt{sum}(X,Y) = \sum_{x<X,y<Y} \mathtt{image}(x,y)$$

$$\mathtt{sqsum}(X,Y) = \sum_{x<X,y<Y} \mathtt{image}(x,y)^2$$

$$\mathtt{tilted}(X,Y) = \sum_{y<Y,abs(x-X+1)\leq Y-y-1} \mathtt{image}(x,y)$$

Figure 5

The equations above clarify how each pixel value is calculated; it is equal to the sum of the value of the source image's corresponding pixel, the value of the pixels above the corresponding pixel, as well as the values of the pixels to the left of the corresponding pixel. The underlying effort of such rapid calculation of weighted pixel sums is in response to the integral image calculations above which make it feasible to compute any rectangular sum with merely four single array calls. The appropriate set, or cascade, of classifiers that is optimal for a high detection rate and a low false positive rate is specific to the object being detected. During the classifier selection process, there are two different data sets that should be used during testing. The first should be a database comprised only of images that do contain the object being detected. The second should be a database comprised only of images that do not contain the object. The results of these two tests in conjunction with chosen threshold values for both detection and false positive rates will help determine which classifiers are optimal for the object detection task at hand. After the classifier training process has been completed, the object detection algorithm can be executed. Figure 6 below shows the software cycle.

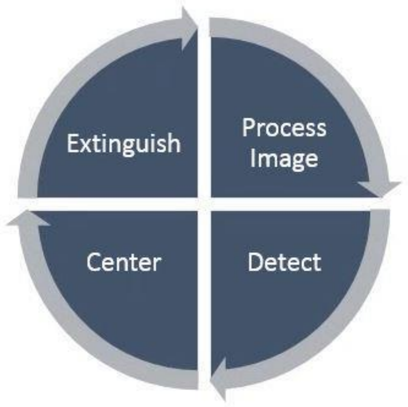


Figure 6

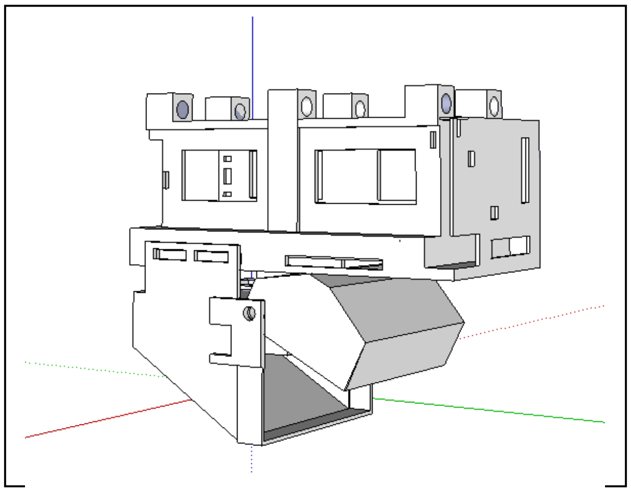
V. Fire Extinguishing System

A. Hardware

The fire extinguishing system will be controlled by a 5V servo. The servo will be controlled by a custom self-designed printed circuit board. Once the quadcopter receives the data that it is directly above the target, a signal will be sent to the custom printed circuit board that indicates to rotate. Once the servo receives the signal, it will retract, opening the contents of the container that is affixed to the quadcopter. The contents inside the container will be a chemical fire suppressant used in many fire extinguishers. The weight of the contents will be approximately 1lb so as to meet our requirements of extinguishing the fire, and also, keeping the quadcopter light enough to still fly. The material within the container will be Sodium Bicarbonate (baking soda). This is used in many BC fire extinguishers, and is the only dry chemical used in large scale for kitchens. Its properties are mostly effective towards grease fires, and small electrical fires. When the chemical heats up, it starts to foam, and effectively smothers the fire.

The hardware aspect of the fire extinguisher release function revolves around a hinge and a servo. This will create a door effect for the container in which the contents will released out of. In order to keep the contents within the enclosure, an electrically controlled device is needed to keep the door like figure closed, and only open when triggered. The servo will rotate the axle upon which the door is attached, which will release the powder inside the casing.

When designing the plastic container that holds the payload, determining the capacity of the container revolved around optimizing multiple constraints. Some key aspects that contribute to our decision are maximum thrust, delivery distance, and size of target. Since this project will be scaled down for sake of demonstration, our target will be much smaller than a real fire. After identifying different power rangers for the thrust of the motors, compared against the weight of the drone and its components, it was determined that 1lb of sodium bicarbonate would be the weight of the payload. It would easily be substantial enough to hit the target, and leave enough room for comfortable lift off and maneuverability. Thus the container was designed to be able to hold up to 1lb of baking soda underneath the drone. See Figure 7 below for the cad drawing of the container.



There are many types of fire extinguishers: dry, wet, even sound. After conducting research, we found Sodium Bicarbonate was the best choice for this project. The ease of access, the performance in fire suppression, and the weight properties all line up for our intended purposes. Baking soda is a crystalline powder that is used for many things including, cleaning, odor neutralization, and even fire extinguishers. Baking soda releases carbon dioxide when heated. Carbon dioxide, while heavier than air, helps in assisting the smother the fire. This makes it a rather effective fire suppressant. While some chemicals are better at extinguishing different types of fires, baking soda has proved to perform well for our intended tea candle or small controlled grease fire. The fire by design will be small enough and of the correct nature, such that our payload of baking soda will possess absolute ability to extinguish the fire.

B. Software

The fire extinguishing system is executed by a 5V servo. The servo's power will be delivered from the battery via the designed Power Distribution Board on the drone. The servo will also be connected to one of the Pixhawk's 6 available auxiliary pins, which will communicated with the servo. Once the drone's software detects that it is above a fire, it will send a signal to the Pixhawk which will rotate the servo until the payload is released. The process of searching for a fire, identifying a fire, moving towards the fire, and hovering over the fire while the fire suppression is released, requires a well-defined state machine so that the drone system is able to complete the missions regardless of conditions. A diagram of the state machine and descriptions of each state are as follows:

Moving towards the center of the fire will occur once the fire is within the camera's vision. This first change that occurs is changing the state of the drone system from Auto mode to Guided mode. In this mode, the drone system is given a GPS coordinate that it moves toward, and once at the coordinate, hovers indefinitely. The Odroid XU4 will estimate a location based on the fire's location within the selected frame from the camera's video feed. Then it will send data to the Mission Planner, which will create a waypoint for the drone system based on the estimated fire location. The waypoints require the data sent as latitude and longitude, and also takes in the length of time to spent hovering at the current waypoint before moving to the next state. The drone system will continually estimate the location until the fire is located in the center of the camera's field of view.

Once the fire is in the center, instead of staying in Guided mode and waiting at a waypoint for X amount of seconds, the drone system will switch into Loiter mode. This mode requires an input deciding the length of time it will spend hovering at the height it enters Loiter mode at. The Odroid XU4 will send the current location to the

Mission Planner, and will then run a script that releases the fire suppressant. Once the suppressant has been released, AND the camera does not detect any temperatures that indicate a fire still exists, the drone system will exit this state.

The final state generates a waypoint based upon the drone's coordinates upon initialization. This will complete the mission and the test or demonstration will be complete. See Figure 8 for the state machine diagram.

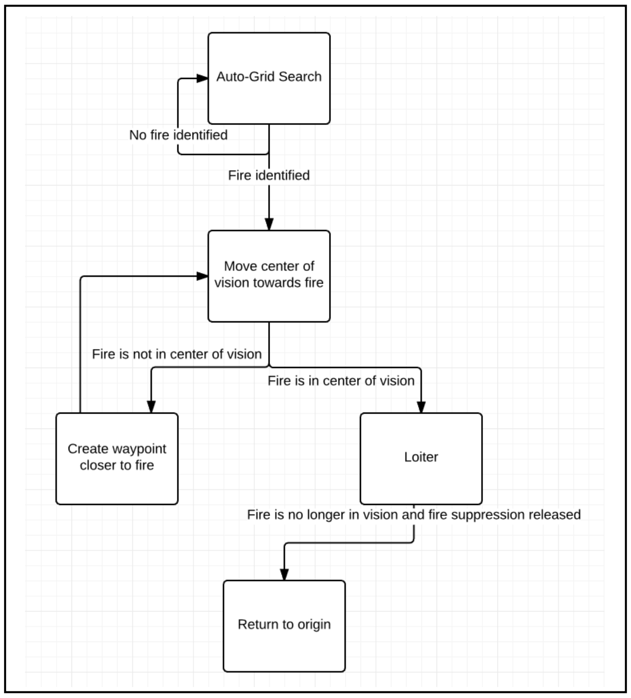

Auto-Grid Search
No fire identified
Fire identified
Move center of vision towards fire
Fire is not in center of vision
Fire is in center of vision
Create waypoint closer to fire
Loiter
Fire is no longer in vision and fire suppression released
Return to origin

Figure 8

VI. Conclusion

FXUAV is a system that aims to prove a concept that unmanned aerial vehicles can assist first responders in hazardous situations such as fires. Using two relatively inexpensive modules, a flight controller and a microcontroller that control the drone, we're able to create an application that can be scaled commercially and used for military purposes as well.

V. Power Distribution Board

A. Design

The Power Distribution System was designed using the Texas Instruments Webench Power Architect. This system is responsible for distributing the power to the Pixhawk, motors, Odroid, and the servo motor from a single power source. There are two main sections of the power distribution system, one section provides 22.2 nominal volts to the motors, and second section drops the voltage down to 5 nominal volts for the Pixhawk, Odroid, and the servo. The section that provides power to the motors uses a current and voltage breakout that provides the Pixhawk with the necessary power information using the Texas Instruments INA169 High-Side Measurement Current Shunt Monitor. The current shunt monitor circuit will be able to sense a current range of 0 to 90 Amps and a voltage range from 0 to 50 Volts. The second section of the Power Distribution System uses the Texas Instruments LM3150 Synchronous Buck Controller. This buck controller is dropping the 22.2 nominal volts battery input to a 5 nominal volts output with a maximum load current of 12 Amps. The components will be laid on a two layer printed circuit board that was acquired from OSHPark. Figure 9 below shows the layout of the Power Distribution System.

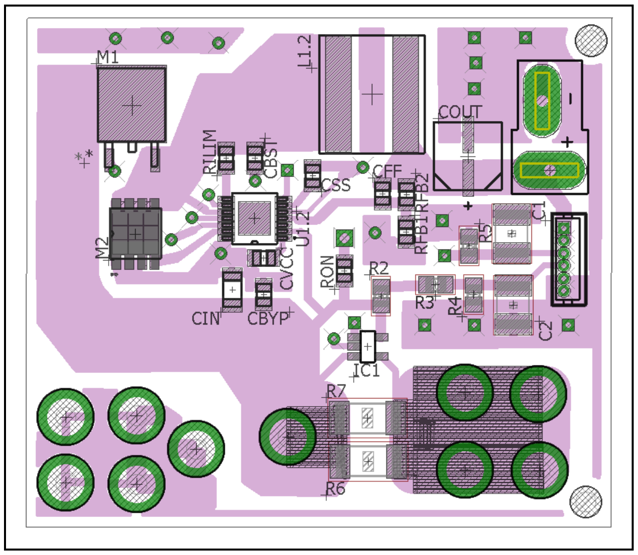


Figure 9

Acknowledgement

The FXUAV team would to thank FLIR Systems for sponsoring the project as well as helping guide the project in the right direction so that it could be a success. A special thanks is also given to Abacus for their contribution of 3D printing materials. All contributors have been invaluable throughout this entire process.

References

[1] Paul Viola, Michael Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. Conference on Computer Vision and Pattern Recognition (CVPR), 2001, pp. 511-518.

[2] "OpenCV: Cascade Classifier Training." OpenCV: Cascade Classifier Training. 18 Dec. 2015. Web. 25 Apr. 2016.

[3] "Advanced MultiCopter Design" ArduPilot. 25 April 2016. Web. 20 March 2016. http://ardupilot.org/copter/docs/advanced-multicopter-design.html

[4] Lorenz Meier, Dominik Honegger and Marc Pollefeys. "PX4: A Node-Based Multithreaded Open Source Robotics Framework for Deeply Embedded Platforms" ICRA (Int. Conf. on Robotics and Automation) 2015. Web. March 10 2016.

[5] Rainer Lienhart and Jochen Maydt. An Extended Set of Haar-like Features for Rapid Object Detection. Submitted to ICIP2002

Luis Brum is a senior Electrical Engineering student at the University of Central Florida. Upon graduating he will be starting his career as a Quality Engineer at Texas Instruments in Dallas Texas.

Greg Kelso is a senior Computer Engineering student at the University of Central Florida. Upon graduating he will be working as a Quartermaster Officer at 20th Special Forces Group in Huntsville Alabama.

Jamie Peck is a senior Computer Engineering student at the University of Central Florida. Upon graduating she will be starting her career as a Software Engineer at Lockheed Martin in Orlando Florida.

Adam Kutchak is a senior Computer Engineering student at the University of Central Florida. Upon graduating he will be starting his career as a software engineer at SpaceX in Space Coast, Florida