Fire Containment Drone

Submitted by:
William Boyd, Zachary Hood, John Lomi,
Chase St. Laurent, Kyle Young

Project Advisor:
Professor William Michalson
Worcester Polytechnic Institute

Submitted to:
Worcester Polytechnic Institute for completion
of Major Qualifying Project

Date:
April 20th, 2017

Abstract

The goal of the fire containment drone project is to create a fire suppression system that can be easily integrated with an Unmanned Aerial Vehicle (UAV) to aid in the containment and control of offshore vessel fires. This system is designed to be an effective and reliable option for land fire-fighting personnel to operate at a safe distance to reduce potential risk and injury.

Table of Contents

Table of Tables

Table of Figures

1 Introduction

Over the course of the year 2014, there were 1,298,000 fires that resulted in 3,275 civilian deaths, 15,775 injuries, and $11.2 billion in property damage. Since 2013, the number of reported fires grew by 4.7% (NFPA, 2016). In the United States, a fire department responds to a fire every 24 seconds (NFPA, 2016). Firefighters are the first line of defense when suppressing fires and almost every situation they encounter is different. One specific situation that often adds increased risk to firefighters is a marine vessel fire. Vessels that are docked at bay or out on the water provide space and accessibility challenges, increasing the difficulty of the firefighting tactics.

The recent innovation and technological influence of robots have created opportunities to make many of the tasks that endanger the men and women who risk their lives everyday easier and safer. Robotic applications are becoming more common in highly hazardous environments in order to reduce any potential risk to humans. One example is the creation of unmanned vehicles, both ground and aerial, which have provided safer alternatives to many high risk situations. Specifically, unmanned aerial vehicles (UAVs) offer a different approach to traditional firefighting techniques. UAVs could potentially offer faster response time, decreased risk to firefighters and a “bird's eye view” of the scenario at hand.

The goal of the Fire Containment Drone project is to develop a UAV in order to aid in the containment and control of offshore vessel fires. To accomplish this goal, a system must be designed and implemented such that it provides a reliable and effective solution to this problem while remaining easy to use by fire personnel.

In order for the system to be successful, a set of requirements must be met. These requirements can be broken down into each of the three key disciplines of robotics engineering: mechanical engineering, electrical and computer engineering, and computer science. In addition, a general requirement is to select an effective fire retardant solution that is appropriate to use on vessels.

There are four main mechanical system requirements. The first is a drone platform capable of carrying a system payload that contains both the fire retardant solution and spray system. Secondly, a lightweight and durable container must be chosen to hold the fire retardant solution. Next, a mechanism to transport the fire retardant solution from a holding container to a nozzle system. This mechanism must provide the appropriate functionality similar to that of a fire hose to provide adequate pressure to be able to attack the fire at a distance that provides safe operation of the drone platform. Finally, a spray nozzle design must be created to produce a spray pattern that provides effective fire suppression.

The main electrical requirement is a custom designed printed circuit board (PCB) to control the electrical power distribution for all subsystems. A PCB and microprocessor will need to work together to control the on-board mechanism, sensors, and signals passed between the multiple devices. In addition, a camera system should be present on the system to allow for video streaming capabilities. Finally, an effective number of communications signals must be implemented to control communications between the drone system and base computer.

The software of the fire containment drone must fully control the drone subsystem. A primary requirement includes the use of a Graphical User Interface (GUI) to provide a user-friendly interface that can provide accurate data and allow a user to perform all the necessary actions to control the system from a distance. For the GUI to be successful, two additional requirements must be implemented. The first is background software to send and receive messages between the drone platform and base computer. The second requirement to compliment the GUI is a video stream from the on-board camera system to the base computer to provide the pilot and operational personnel with a first person view. This is very important for the pilot of the drone to know the location of the drone relative to the environment. In addition, vessel fires have the potential to be very difficult for land based firefighters to access. A drone platform streaming video gives the option to assess a situation that may not be accessible otherwise. A final requirement to aid to aid fire-fighting operations is the implementation of a fire detection functionality to help locate the fire via the drone's camera system.

The contents of this paper will first explore the background of fire research and containment as well as its impact in maritime settings. In addition, background research will be conducted in the use of drone technology. After the background has been established, the paper will examine many different potential options to meet the requirements needed to have an effective solution. Mechanical concepts such as spray patterns and nozzles, pumps, tubing, tanks, and the overall number of degrees of freedom of the fire suppression system will be compared and evaluated. For electrical and computer engineering, concepts such as image sensing via camera sensors, system power distribution, and the wireless communications between the UAV and the base computer will all be discussed. In addition, the paper will cover key components in the software used to run and control the UAV, its many subsystems, and the function of the base computer.

Finally, using research, modeling, and various testing methods, the best options to fulfill each of the above stated requirements were chosen by the team to construct a reliable effective system. The proposed solution was prototyped and tested. The results, analysis and conclusions are discussed in the later portion of the report.

2 Background and Literature Review

2.1 Fire Basics

Fire is the process of oxidation, or the combining oxygen with another substance, happening at a very fast rate. Fire releases energy that produces both heat and light and there are three components to every fire: fuel, heat, and oxygen shown in Figure 2.1. All three components must be present to have a fire (NFPA, 2016). Fire will continue to burn and spread while the three of these components are available. The life of a fire is broken up into four stages. The first is the starting ignition when fuel, heat, and oxygen join together to start a chemical reaction. The second stage is the growth of a fire. This is when the original ignition produces enough heat to ignite additional fuel. The third stage is when the fire has fully developed. At this stage, fire has spread over much if not all of the available fuel and oxygen is being rapidly consumed. This stage results in the most heat damage. The final stage is the decay of a fire. Once all the fuel has been consumed, temperature decreases, lowering the intensity of the fire.

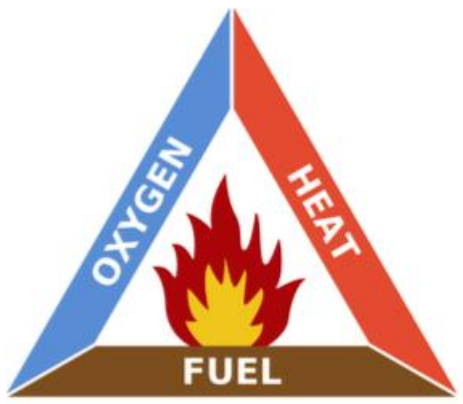


Figure 2.1: Fire Triangle

The strategy of fire suppression is to remove at least one of the three sides of the fire triangle. This consists of either cooling the burning material, excluding oxygen, removal of the fuel sources, or breaking the chemical reaction (NFPA, 2016). The suppression method used is based on the classification of the fire which is based on the fuel type. Class A fires consist of ordinary combustible materials like wood, cloth, paper, rubber, and plastics. Water is the most effective agent in suppressing Class A fires. Class B fires consist of flammable liquids and combustible liquids such as tars, oils, alcohols, and flammable gases. Foam is the most effective agent in suppressing these fires. Class C fires are anything that would be considered class A or B with the addition of energized electrical equipment. Carbon dioxide and dry chemical agents are the most effective suppression technique for this type of fire however, water can be dangerous due to its conductivity. Class D fires consist of combustible metals such as magnesium, titanium, and sodium among others. These metals usually burn at extreme heat making them difficult to suppress with water. Class D fires are often suppressed with special powders based on sodium chloride and sand. Finally, Class K involves fires in cooking applications that involve cooking substances such as vegetable or animal oils and fats (NFPA, 2016).

2.2 Marine Vessel Fire Fighting

As long as there is fuel, oxygen, and heat, fire can happen anywhere. Although most fires take place on land, whether structural or outside, the small number of fires that happen on marine vessels both docked and out at sea can be more negatively impactful to people and the environment. Marine vessel firefighting presents all new challenges to firefighters unique to those for a land fire. Increased difficulty to accessibility, the challenge of a moving structure fire, and higher pollution impacts are just a few of these challenges. Firefighters must adapt to the added risk presented in these situations.

2.2.1 Marine Vessel Fire Statistics

In 2014, the United States Coast Guard reported 4,064 accidents that involved 2,678 injuries, 610 deaths, and approximately $39 million in damages to property as a result of recreational boating accidents (Coast Guard, 2014). Of those accidents, 263 were due to fire-related accidents leading to 129 injuries and 5 deaths with over $12 million in property damage (Coast guard, 2014). Although they only account for less than one percent of recreational boating accidents, fires on marine vessels can be very difficult to contain and have even led to loss of life. Statistics for marine vessel fires are relatively consistent yearly from 2010-2014 as seen in Table 2.1 below.

Year	Accidents	Injuries	Deaths	Damage ($)
2014	263	129	5	12,798,427
2013	226	107	0	12,586,601
2012	277	106	7	13,849,847
2011	221	109	7	28,551,173
2010	251	104	2	11,764,352

Table 2.1: Marine Vessel Fires 2010-2014 (Coast Guard, 2014)

Table 2.1 shows that the most number of accidents happened in 2012 closely followed by 2014. In the five-year span, there have been 21 deaths recorded and $80 million in property damage. Although marine vessel fires are not leading in any category, the dangers presented to not only the occupants of the vessel, but for fire personnel and the environment, are very high.

2.2.2 Firefighting Strategy and Tactics

The National Fire Protection Agency (NFPA) has developed a guideline for land based fire departments that respond to marine vessel fires. This guideline can be found in the NFPA 1405 code (NFPA, 2016). According to the NFPA, the U.S Coast Guard is interested in helping in fire prevention and firefighting tactics, however local authorities are principally responsible for maintaining necessary fire-fighting capabilities in U.S ports and harbors. The Coast Guard will provide assistance without conveying the impression that the Coast Guard will relieve local fire authorities of their responsibilities (NFPA, 2016). The U.S Coast Guard will usually help in coordinating firefighting operations for marine fires, however they do not carry enough equipment for fire-fighting capabilities. The U.S Coast Guard determines anchorage locations, which are locations that often provide isolation from other vessels. Firefighting operations can be performed in these locations by local fire departments providing less exposure problems and risk than there would be present in a populated area. (NFPA, 2016).

The general tactics for fire-fighting operations revolve around confinement, preventing the spread of fire, prevention of endangerment to people, and protection of exposures. Environmental impact is also a great influencer in decision making strategies when handling fire-fighting operations. Water pollution due to the high potential of hazardous materials on a marine vessel is very significant. Pollution impacts from firefighting operations can easily exceed the impact of the fire itself (NFPA, 2016). According to the Florida Keys Area Committee's Marine Firefighting Plan, when a vessel and its cargo are deemed a total loss, it may be best to sink it in an area where environmental impact is minimized (Florida Keys, 2012).

Every vessel response should initially be treated as a hazardous materials incident. Hazardous materials response units should always respond and/or be immediately available to a marine vessel fire. A command post (CP) must be set up to direct operations during a marine vessel fire. If the vessel on fire is far from land, a floating CP can be established as close to the scene as possible without endangering CP personnel (Florida Keys, 2012).

The two most common fire suppression methods used in marine vessel fires are foam and water. When attacking with foam, the fire-fighting vessel should be moved where the foam is being sprayed downwind, and the attack on the fire should only start if enough foam is readily available to suppress the fire as re-ignition is highly possible (NFPA, 2016). If attacking with water, the area must remain accessible and dewatering capabilities must be established to keep stability of the vessel in the water. This is more prominent with smaller vessels.

2.2.3 Fire Fighting Risk

In addition to the generic risks associated with fighting fire, firefighters face additional risk when battling a marine vessel fire. One of the most different aspects of marine firefighting versus land firefighting is the effect of tides and currents. Tides can create strain on weakened parts of vessel creating stability issues for firefighters trying to board a vessel. High tides and fast currents can also make attaching anything such as a ladder or anchor system very difficult (NFPA, 2016). Furthermore, as with any fire situation, firefighters must use full protective gear including a personal floatation device. Therefore, falling into the water is a huge risk and in any rescue situation, there needs to be a way of detecting and rescuing fire personnel who fall into the water. Small rescue boats are essential as most docks and piers can be too high compared to water level and most fire-fighting vessels are too large to provide fast rescue operations.

Shipyards and dry-docks are where most docked vessel fires occur due to the welding, burning, and grinder jobs that happen in these areas. In these types of environments, many vessels are only accessible by one gangway. This can make firefighting operations confined and limited due to space. In addition, floating piers have weight limits that can easily be reached when multiple fire personnel in full gear are on them (NFPA, 2016). According to the NFPA, falls are the number one hazard for firefighters in a marine dock environment due to an increase slip ratio (NFPA, 2016). Other risks that are present to firefighters in marine vessel fires include limited air quality below deck in a vessel as well as hazardous materials and/or flammable liquids stored in compartments on a vessel. These are just some of the many increased risks during marine fire operations and in no way represent a complete list of the risks presented to firefighters. Additional risk factors may be presented based on the situation.

2.3 Fire Suppression Techniques

2.3.1 Carbon Dioxide, Water and Foam

There are many different types of fire suppressant techniques used. Of these, carbon dioxide, water, and foam are the most common. Carbon Dioxide is a very common user-friendly suppressant as it can be easily stored in a pressurized handheld extinguisher. A carbon dioxide (CO2) extinguisher is designed to apply carbon dioxide directly to a fire that could occur in an area or space that essentially has no enclosure surrounding it (NFPA 12, 2015). Since carbon dioxide is non corrosive, non-damaging, and leaves no residue, CO2 is an ideal fire suppressant as it can spread to all parts of a fire (NFPA 12, 2015) In addition, as a gas, it will not conduct electricity and can be used on electrical hazards. Carbon dioxide is effective because it creates a blanket preventing the fire from receiving oxygen, therefore eliminating one side of the fire triangle.

Foam is made of an aggregate of air-filled bubbles used to form a cohesive floating blanket on flammable and combustible liquids (NPFA 11, 2016). It suppresses a fire by excluding air and cooling fuel thus helping to prevent the spread and re-ignition of the fire. Foam is the only permanent agent used for spills or tanks of burning flammable or combustible liquids

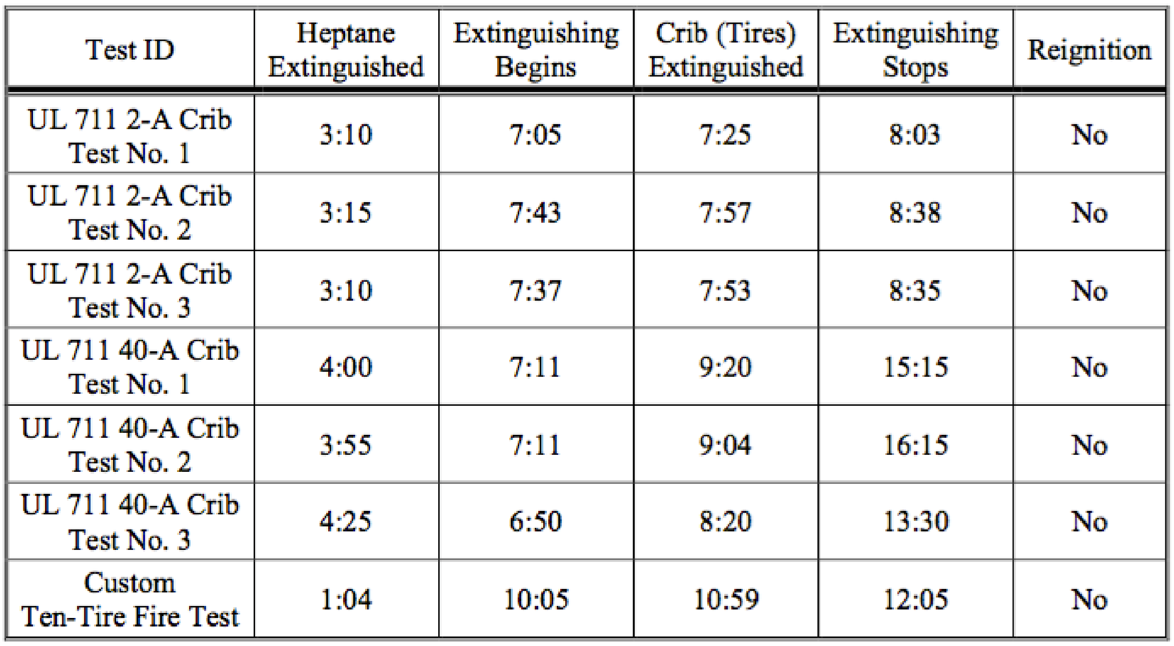
(NFPA 11, 2016). There are many different types of foam used for different firefighting applications. Foam can also be used as fire prevention as it can be applied to flammable liquids that have not yet been ignited in order to create a blanket, preventing ignition from another source (NFPA 11, 2016).

Finally, water is the most widely used fire suppressant due to its ability to attack all three sides of the fire triangle. Water reduces the heat of a fire and while doing so, displaces air from fire, cutting off a fire's oxygen supply. Additionally, water can be used to directly attack the fuel source suppressing the supply. The abundance of water is another reason which makes it so successful. Most major cities and towns have developed and adapted a fire hydrant system to be able to deliver water in abundance to anywhere, allowing firefighters easy access to fire suppressant.

2.3.2 FireIce

While carbon dioxide, water, and foam are the common fire suppressants used by firefighting services, other products have been developed over the years as alternatives. One these alternatives is GelTech Solutions' FireIce. FireIce is an eco-friendly polymer that mixes with water to create a fire suppression substance that not only extinguishes fires at a fast rate, but stops fires from starting again. The mixture is biodegradable making it eco-friendly and non-conductive, allowing the suppressant to be used on electrical and utility equipment. In addition, FireIce has innovated a new way of providing home fire protection and extinguishing forest fires (GelTech, 2017).

To help fire personnel fight fire, GelTech Solutions has produced equipment for FireIce including a Class-A Fire Extinguisher than has been certified and tested by Underwriters Laboratories (UL). In addition, the Southwest Research Institute in San Antonio, Texas conducted a fire extinguisher performance evaluation using FireIce in accordance with the UL 711 wood crib fire test (Weyandt & Janssens, 2008). The evaluation was done on Class 2-A and 40-A cribs as well as a ten-tire fire to record the extinguish time and reignition if applicable. The results of the test are shown in Table 2.2 below.



Test ID	Heptane Extinguished	Extinguishing Begins	Crib (Tires) Extinguished	Extinguishing Stops	Reignition
UL 711 2-A Crib Test No. 1	3:10	7:05	7:25	8:03	No
UL 711 2-A Crib Test No. 2	3:15	7:43	7:57	8:38	No
UL 711 2-A Crib Test No. 3	3:10	7:37	7:53	8:35	No
UL 711 40-A Crib Test No. 1	4:00	7:11	9:20	15:15	No
UL 711 40-A Crib Test No. 2	3:55	7:11	9:04	16:15	No
UL 711 40-A Crib Test No. 3	4:25	6:50	8:20	13:30	No
Custom Ten-Tire Fire Test	1:04	10:05	10:59	12:05	No

Table 2.2: Southwest Research Institute FireIce Test Results (Weyandt & Janssens, 2008)

As one can see by the fourth and the rightmost column in Table 2.2, FireIce successfully extinguished and prevented reignition in all tests. On average, FireIce extinguished the fire in about 17 seconds for the 2-A crib test and 110 seconds for the 40-A crib test. The test concluded that FireIce successfully meets the UL standards for rating and testing of fire extinguishers. FireIce has recently been approved for use by the United States Forest Service and currently being used by firefighters, first responders, and military personnel (GelTech, 2017).

2.4 Drone Technology and Applications

Unmanned Aerial Vehicles (UAVs), also known as drones, are aircraft that do not have a pilot onboard and are either controlled remotely or are autonomous. Drones have many military and commercial applications. There are two types of UAVs, fixed-wing and rotary-wing. Both designs have their advantages and disadvantages and are used in different applications (Floreano & Wood, 2015).

2.4.1 Types of Drones

Fixed-wing drones operate similar to airplanes where the roll, pitch, and yaw are controlled by ailerons, elevators, and a rudder respectively, which are found on rigid wings. Roll is the rotation about the x-axis of the center of gravity (COG) of the drone, pitch is the rotation about the y-axis of the COG of the drone, and yaw is the rotation about the z-axis of the COG of the drone. The thrust is typically provided by a propeller or some other form of forward propulsion and lift is generated by the wings of the drone. Refer to Figure 2.2 for a visual representation. his type of drone is good for surveying areas, mapping regions, moving large payloads, or any other application that that requires a large distance to travel or area to cover (Peterson, Neilan, Rubenstein, & Toribio, 2016).

2.4.2 Drones and Fire

Fire impacts the drone through changing air currents that cause turbulence and potential heat damage to the drone. Turbulence is generated by a fire because the fire's hot air creates airflow up towards the drone (SFPE Handbook, 2016). Drones are traditionally lightweight and fragile, so turbulence can have a large impact on how well they can navigate. The heat from the fire can also cause damage to the mechanical and electrical components if they are overexposed. Most drones have a frame and body made of plastic as plastic provides a strong enough base and is lightweight. However, plastic has a lower melting point in comparison to other materials like metals and this can cause issues if the drone stays around the fire for too long (Peterson et al., 2016).

An important characteristic to take into consideration when fighting a fire with a drone is the amount of heat that it is dissipating. The heat dissipated by the fire is influenced by many factors such as type of heat transfer, distance, and time. The types of heat transfer are convection, conduction, and radiation. The drone will be flying around the fire so there will be no heat transfer via conduction as there will be no solid contacts made with objects producing heat. In an open-air environment, convection is negligible if the drone does not fly directly over the fire. Therefore, the primary heat transfer influencing the heat experienced by a multi-rotor drone is radiation (Peterson et al., 2016).

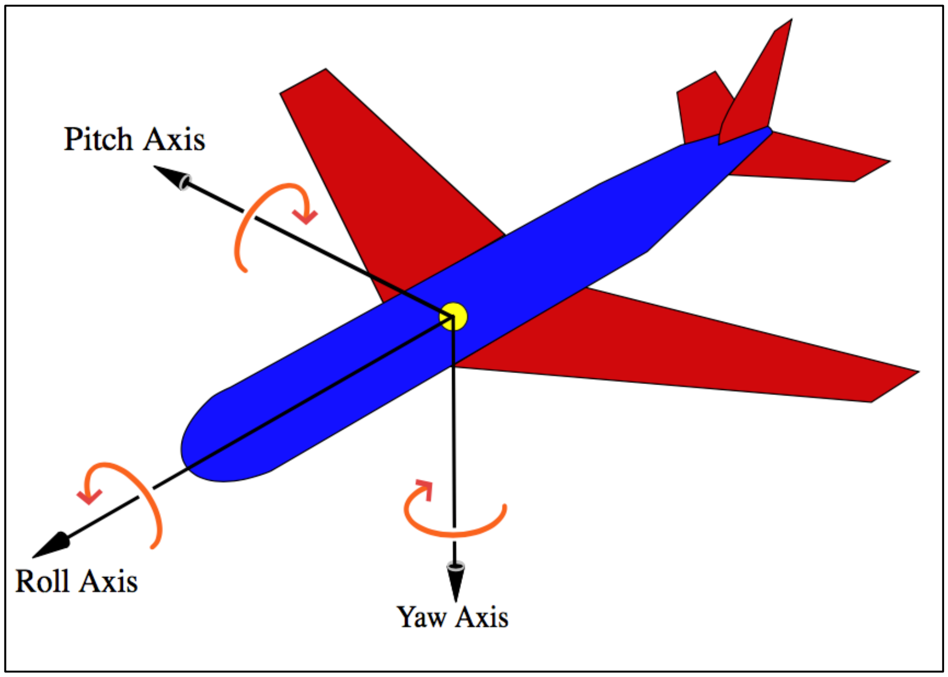


Figure 2.2: Axis of Aerial Motion

Rotary-wing, or multi-rotor drones have more than two rotors mounted on the drone which provide upward thrust and generate lift. The roll, pitch, and yaw of a multi-rotor drone are controlled through changing the speeds of individual propellers. Typically, rotors are used in pairs and will rotate complementary to other pairs of rotors to maintain stability and control. This type of drone is good for applications that require the drone to hover, stay in a fixed area, or applications that require precision (Peterson et al., 2016). Figure 2.3 below shows some of the possible configurations of multi-rotor drones.

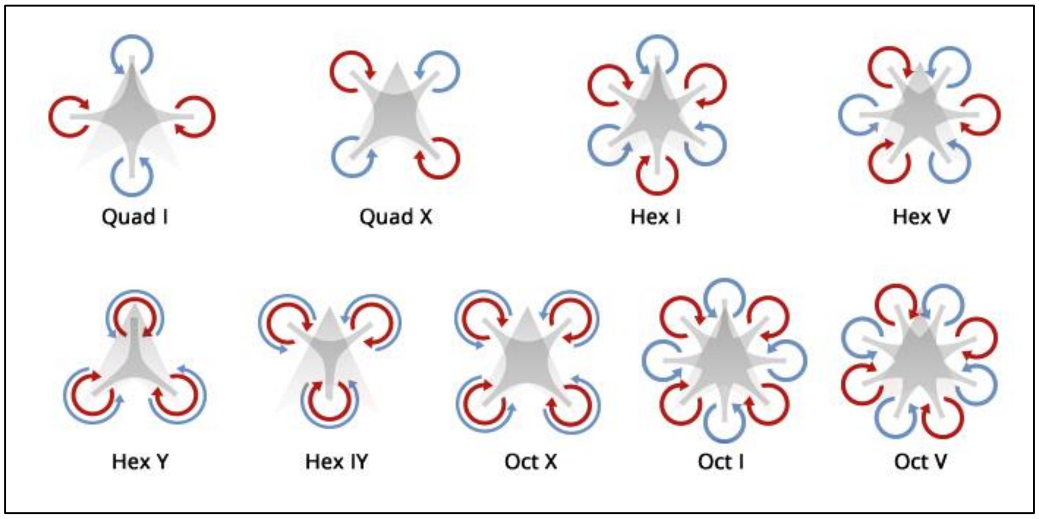


Figure 2.3: Multi-rotor Drone Configurations

2.5 Pre-Determined Design Elements

Due to the previous project and research done with fire containment drone by last year's team, a few system designs were already predetermined. The first of these specifications was the drone platform. Through previous research and current accessibility to an available drone platform, the DJI S1000 multirotor drone, controlled through a Pixhawk 4, was used for the testing of the fire suppressant system. The S1000 has a payload of approximately 11 pounds with a flight speed of roughly 40 miles per hour (mph). With a fully charged battery, the drone had a flight time of around 15 minutes when carrying the previous fire suppression system (MQP, 2016). Last year's team also determined the fire suppressant, FireIce, which was used in the current system. The team decided to continue using FireIce due to its ability to handle Class A, B, and C fires, which are the most common types of fires on boats (MQP, 2016). Additionally, FireIce is non-conductive making fighting electrical fires on marine vessels safer, and since it is eco-friendly, it has no negative environmental impacts. FireIce has also been proven to prevent reignition through testing. Furthermore, the team already had access to the fire suppressant making the testing low cost and therefore increasing the feasibility of using FireIce.

2.6 Mechanical Background

One of the most integral components of the Fire Containment drone is the mechanical system which has the goal of propelling FireIce from the drone to the flaming boat. Factors that influenced the end design are the required distance to propel the FireIce, the minimum spray coverage desired, and desired user intractability. A limiting factor in many design considerations was the maximum weight capacity, where the DJI S1000 drone is only capable of carrying 24.25 lbs. The key components of the mechanical system that were researched and implemented were the spraying system, the degrees of freedom of the system, the pumping system, and the FireIce container.

Various other factors were assessed and estimated in conjunction with these objectives such as cost, size requirements, weight requirements, feasibility, and complexity. Size is important because the drone has a limited space underneath its structure to host the system and it was important to keep the weight of components in mind due to the payload capacity as previously mentioned. Feasibility and complexity were assessed for each component due to the tools and time, respectively, available to the team. All of these factors were considered for all the design alternatives and were input into a respective decision matrix for each component of the mechanical system.

2.6.1 Spray Pattern and Requirements

The main objective of the spray pattern device was to atomize the FireIce to the best possible degree before applied to its target. It is important to note that the process of atomization must be done in a relatively controlled manner as outside forces such as wind can cause the spray path to miss the intended target if the FireIce droplets become too small too soon (Grant, Brenton, & Drysdale, 2000). It is also beneficial to optimize the longitudinal coverage of the spray so as to ensure that more of the fire is contained during spraying. This process involved preliminary calculations to estimate the system output, and testing was done to confirm these estimates. Multiple conceptual ideas were surmised to accomplish a satisfactory spray pattern. These ideas were researched and assessed on multiple factors that were collectively input into a decision matrix and evaluated quantitatively.

Fluidic Oscillator

The first option was a fluidic oscillator. A fluidic oscillator is a nozzle with an inner chamber that causes pressure imbalances which in turn force the fluid exiting the nozzle to oscillate back and forth or up and down. The main benefit of this is that it does not require any external sources of energy to generate its spray pattern. The spray pattern is driven by the flow of the fluid itself in the nozzle chambers (Kim, 2011). The main drawback with this idea, however, is that it is difficult to create and requires many iterations to perfect. It would be low cost because 3D printed nozzle parts could be made for testing and a single final product would be manufactured. It would also be of minimal additional size as the spray pattern inducer is contained within the nozzle itself. Overall, it would be able to effectively atomize the FireIce solution due to the rapid change in fluid direction.

Pneumatic Actuation

A pneumatically induced system would feature a piston attached to a pivot that is fixed to the nozzle, which would rapidly oscillate the nozzle position. The main drawback and major factor affecting the feasibility of pneumatic actuation is that the compressor may not be able to keep up with the constant actuation of the piston in order to generate the oscillation frequency desired.

Electromechanical Actuation

This design would implement a motor that would pivot the nozzle on a fixed range of motion. This design could implement a gearbox depending on the speed and torque that this system would require. This would work well for atomization as it is possible to control the rotation through a feedback loop and make a smooth sweeping motion for a set range of motion, however, it may not be able to oscillate at a desirable frequency.

Impact Sprinkler

An impact sprinkler is a device typically used on lawns to create a spray pattern on the grass. This system works by having a hammer hitting the stream which in turn causes the stream to shift positions in one direction around its axis. Eventually this action causes it to reset to its initial angle orientation where it repeats the cycle. This would work moderately well for atomization, but overall it doesn't seem like a feasible idea because it would alter the flight pattern and possibly influence the drone's alignment with the fire.

Multiple Exit Nozzle

Multiple exit nozzles increase coverage by ejecting the spray in various directions in which the independent nozzles are oriented. The main drawback of this design is that the additional pressure generated will not better the system more than a basic fanned nozzle could, and the design would require more space.

Piezoelectric and Vibration Motors

A vibration induced spray pattern would use a piezoelectric motor or vibration motor to induce an oscillation pattern. The main drawback with this design is that it may be difficult to generate enough force out of the piezo motor or vibration motor to actuate the nozzle as desired.

Fanned Nozzle

A fanned nozzle is a typical nozzle with a spray angular degree greater than 0. This causes the stream to thin out and gradually spread horizontally as it travels. The only drawback with this is that a high degree of angling could cause the spray to be too fanned out and a lot of FireIce would be wasted.

Solid Stream Nozzle

A solid stream nozzle keeps the fluids flow as it exits the nozzle uniform. By using this nozzle, the FireIce will not atomize as well, but it will be able to achieve greater distances and be disturbed less by wind fluctuations. This is the most reliable option for propelling the FireIce to the target fire.

Spray Pattern Decision

A decision matrix was generated in order to assess each option as seen in Table 2.3 below. The components were rated from very low to very high. These were opinion based and averaged ratings based upon the cumulative research done on each design idea.

Solution	Power Requirements	Cost	Size Req.	Complexity or Risk	Atomization	Feasibility	Score
Fluidic Oscillator	No Additional	Low	Low	Med/High	High	Med	**25**
Pneumatically Induced	Pneumatics	High	High	Med	Med/High	Low	**11**
Electro-mechanically Induced	Servo	Low/Med	Med/High	Low	Med/High	Low/Med	**17**
Impact Sprinkler	No Additional	Low	Med	High	Med	Low	**17**
Multiple Nozzles	No Additional	Low/Med	Low/Med	Low	Low	Med	**23**
Vibration Induced	Piezo/Motor	Low	Low	Med	Med	Med	**24**
Fanned Nozzle	No Additional	Low	Low	Low	Med	High	**28**
Solid Stream	No Additional	Low	Low	Low	Low	High	**26**

Table 2.3: Spray Pattern Decision Matrix

From this matrix our team was able to conclude that the most viable options to pursue were the fanned nozzle, the solid stream nozzle, and the fluidic oscillator. The previous MQP group had also come to similar conclusions, where they decided upon a solid stream and an angled flat spray, confirming that our assessment of these options are supported by previous research as well.

2.6.2 Degrees of Freedom

The degrees of freedom (DoF) of the mechanical system are important to the system as they define the level of control and complexity the system has. The main goal of introducing additional degrees of freedom is to allow for the user to control the direction of the spraying system. There were three independent mechanical system layouts assessed: 1 and 2 DoF at the nozzle, 1 and 2 DoF of the entire mechanical system, and 0 DoF of the entire system and nozzle. The goal of doing this was to find the best and most practical approach of targeting the fire with the spray system. The options discussed are shown in Table 2.4 below.

Number of DoF	Location of DoF 1	Location of DoF 2
0 DoF	N/A	N/A
1 DoF	At the Nozzle, Longitudinal	N/A
1 DoF	At the Nozzle, Latitudinal	N/A
1 DoF	Entire System, Longitudinal	N/A
1 DoF	Entire System, Latitudinal	N/A
2 DoF	At the Nozzle, Longitudinal	At the Nozzle, Latitudinal
2 DoF	Entire System, Longitudinal	At the Nozzle, Latitudinal
2 DoF	Entire System, Longitudinal	At the Nozzle, Longitudinal
2 DoF	Entire System, Latitudinal	At the Nozzle, Latitudinal
2 DoF	Entire System, Latitudinal	At the Nozzle, Longitudinal

Table 2.4: DoF Considerations

Based upon the options presented above, the team decided at least one DoF was required for the system as it would be necessary to control the path of the FireIce by either pitching it upwards or downwards, eliminating the option for a 0 DoF system. It was also quickly decided by the team that there was no need for any latitudinal translation or rotation of the system, as the spray pattern would accomplish this. This eliminated the options involving any latitude rotation as well as all of the 2 DoF considerations. Finally, the team decided on the 1 DoF at the Nozzle with longitudinal rotation. This configuration allows the operator to reorient the pitch of the nozzle in an attempt to better direct the flow of FireIce to its target.

Driving Mechanism

In designing this subsystem, the driving mechanism to control the motion of the joint was analyzed. The considerations were a stepper motor, a high powered servo, and a DC motor both with and without an encoder. The team quickly decided against the use of an encoder with the DC motor because it would require hardware interrupts, and the team deemed this would be too unreliable. These driving mechanisms can be seen in Table 2.5 below.

Driving Mechanism	Voltage Requirement	Software Restrictions	Other Electrical Requirements	Stall Torque (Nm)	Weight (lbs.)
Stepper Motor	12 V	Blocking	Driver Chip	2.938	1.109
PWM High Powered Servo	5 V	Non-Blocking	N/A	1.520	0.132
DC Motor	12 V	Blocking	H-Bridge	1.765	0.518

Table 2.5: Driving Mechanisms

The above table highlights the strengths and weaknesses of each driving mechanism option. The team decided that the stepper motor was not as feasible of an option due to the fact that it required a driver chip and had excessive weight, despite the significantly greater stall torque and precision motion. The team also decided against the PWM servo module because while its stall torque was adequate, the team preferred to have a higher stall torque as well as a more robust motor. In the end, our team chose the DC motor, but instead of using an encoder to measure its rotation, we decided to use a potentiometer.

2.6.3 Pumping System

Determining a pump for the mechanical system was very important because the system relies on it to pressurize FireIce, a relatively viscous substance, in order to propel it a reasonable distance. Based upon the previous team's report, the pressure needed at the exit is 100 psi in order to get an adequate suction and flow from the FireIce substance (Peterson et al., 2016). However, their team was only able to propel the FireIce solution 19 feet, only 4.5 feet over their required safe distance from the fire. If a gust of wind were to alter the drones position to somewhere within the unsafe operating range, it would be a problem as components could melt. Our team decided to revisit the pumping system and create a new system for this application in order to maximize our spraying distance so that there is a greater safety for the drone. To do this, multiple pumping mechanisms were analyzed and a decision matrix was created to decide upon the top viable candidates for pumps.

Gear Pumps

The first solution for a pump system was the gear pump. There are two main classifications of gear pumps: external and internal. An external gear pump is configured with a gear ratio of 1:1 consisting of a driver and a driven gear. In this design, fluid flows from an inlet and around the gears in the direction away from where the gears make contact. During the flow around the gears, the fluid is translated to an outlet. The gears, at a fixed rotational velocity, create suction and pressurize the system through its motion.

Internal gear pumps consist of an internal gear, similar to a sun in a planetary gear system, that rotates and meshes with a gear that revolves around it with an offset center, similar to a ring gear. One variation of this includes a chamber between the natural gap in between the driver and the ring gear. These configurations are capable of generating higher suction. The variation with a chamber is capable of pumping viscous substances.

These systems require an external casing surrounding the gears that should be fitted to the inlet, gears, and outlet to create the most optimized system. All three systems would require relatively low size requirements as the gears and structure can be made moderately compact. By adjusting the drive motor speed, one can adjust the rotational velocity of the gears and in turn increase or decrease the pressurization of the fluid in the gear pump.

Centrifugal Pump

This type of pump has a rotational element, the impeller, that is operated via an electric motor. The fluid enters an inlet to the impeller and as it spins, the fluid is accelerated radially outwards from the center, exiting from an outlet. This design would require that fluid be present for the impeller so as not to create a dry suction which can harm the system. Centrifugal pumps come in many sizes, but, based upon our team’s research, those suitable for application on the drone would require excessive space in the system and would exceed the drones critical weight.

Diaphragm Pump

A diaphragm pump is a positive displacement pump that uses a diaphragm and valves to create suction. Diaphragm pumps can be up to 97% efficient and are good at self-priming. They are able to handle highly viscous liquids and can have high discharge pressures. This type of pump would require a moderate amount of space on the drone and a moderate amount of additional weight. This was the pump of choice by the previous team and this design was taken into consideration and compared to the other possibilities during the testing phase.

Rotary Vane Pump

A rotary vane pump has a rotating element within a cylindrical casing that hosts an inlet on one side and an outlet on the opposite side. The rotating element has a series of cutout slits or chambers which hold a sliding plate that is either pushed away from the rotating element by a spring or pneumatics. These plates are pushed in and out through contact with its container walls. As the rotating element rotates, the chamber plates move in and out and on one arc of the system, transfers the fluid from the inlet to the outlet. This system would require an external motor in order to power the rotating element.

Screw Pump

A screw pump is a positive displacement pump that uses one to many screws to move a fluid along their axes and is great with highly viscous fluids This type of pump has the advantage of moving without turbulence, an ability to pump high viscosities without losing flow rate, and the ability to operate despite changes in the pressure. The major downfall is its typically tremendous size and weight.

Pneumatic Hand Pump

This design takes the properties of a hand pump design, but instead actuates it with pneumatics. The hand pump design uses a plate inside the tank that pushes against the volume of solution causing the pressure to increase and in turn cause the fluid to flow through the system. The issues with this design the additional spatial and weight requirements for pneumatics, the variances in output pressure, and an inability to control amount of FireIce ejected after activation.

Pumping System Decision

A decision matrix was created for the pump options in order to properly assess each solution for the previously stated categories, the cost, feasibility, weight requirements, size requirements, and power requirements were all taken into consideration. From this matrix and these categories, multiple solutions were assessed and a quantitative ranking operation was implemented to determine which options were viable to pursue, as shown in Table 2.6 below. The components were rated from very low to very high. These were opinion based and averaged ratings based upon the cumulative research done on each design consideration.

Solution	Power Requirements	Cost	Size Req.	Complexity or Risk	Pump Ability	Score
Gear Pump: 1 to 1	12V Motor	Med	High	Low	Med	**12**
Geroter	12V Motor	Med	High	Low/Med	Med	**11**
Geroter with Chamber	12V Motor	Med	High	Med	Med	**10**
Centrifugal Pump	12V Motor	Med	Med	Low	High	**16**
Diaphragm Pump	12V Motor	Low/Med	Med	Low	High	**17**
Rotary Vane Pump	12V Motor	Med	High	Med/High	Med	**9**
Screw Pump	12V Motor	High	Med/High	Med	Med	**9**
Pneumatic Pump	Pneumatics	Med	Med/High	Med/High	Low/Med	**9**

Table 2.6: Pump Decision Matrix

From the decision matrix, our team was able to conclude that our most viable pump options were the diaphragm pump, the centrifugal pump, and the gear pump. Upon initial design iterations, the gear pump and centrifugal pumps were quickly dismissed, as they either were too large for our system, weighed too much, or a combination of both. In the end, our team decided upon pursuing diaphragm pumps.

2.6.4 Tubing System

The tubing portion of the mechanical system is responsible for the transfer of the FireIce solution from the tank to the pump and from the pump to the nozzle. Factors to consider when determining which tubing material and dimensions to choose are friction, flexibility, and rigidity.

The frictional coefficient of the material properties for the selected tube is important as the lower the frictional coefficient, the less energy is lost in the system. For the flexibility and rigidity, two options were considered. One option was that the tubing must be flexible so that the nozzle will be able to achieve 1 or 2 degrees of freedom. The other alternative is for if there is to be no degree of freedom at the nozzle, in which case there is no need for flexibility. In this case, the tubing material properties will need to be focused on high rigidity and low flexibility.

Due to the selection of a single degree of freedom at the nozzle, our team decided that it was important to have some flexibility in the tubing. In the end, our team decided that it would be best to have flexible tubing connecting the tank to the pump. From the pump to the nozzle, we decided upon transitioning from flexible tubing to a rigid pipe near the nozzle to better control the fluid flow.

2.6.5 FireIce Containment System

In order to extinguish the boat fire, the design required a method of containing the FireIce. The container had to be lightweight while still able to hold as much FireIce as deemed necessary. Also, the container needed to have entry and exit ports to allow for the insertion of FireIce into the tank and the exit to the nozzle.

The previous team delved into this topic and created 4 alternatives for tanks: a consumer off the shelf (COTS) container, a PVC container, and a custom Trapezoidal container. Based on their findings they decided to pursue the COTS container because it was capable of holding one half-gallon while only weighing 0.631 pounds when empty. When filled, it weighed 4.17 pounds, and since weight is very important in this system, the team determined that this low weight and high volume solution was the best choice.

Our team used this previous analysis to come to our own conclusions, deciding upon creating our own tank that would weigh less than or equal to 4 pounds, reducing the weight this portion of the overall system incurs.

2.7 Electrical Background

2.7.1 Camera System

Having a camera on board the drone will allow the operator of the drone have visual feedback from the drone to the base computer. This will enable the operator to have knowledge of the flight path of the drone during all stages of the flight, and thus enable the operator to make real-time decisions on how to control the drone. Due to the variety in camera systems, the team looked into two main types of cameras, visible light and thermographic.

Thermographic Cameras

Unlike the typical camera, which uses light energy, Thermal cameras operate by heat energy. This heat energy is emitted by all objects such as humans and animals, as well as inorganics like fire and boats. One of the limitations of visible light cameras, as mentioned above, is the difficulty in sensing in the absence of light energy. Because thermal cameras sense using thermal energy instead of visible light, this ceases to be an issue. However, thermal imagers are not without their own limitations. Because thermal energy does not have a visual spectrum by which heat can be seen, thermal cameras must rely on other means in order to display the data that it gathers. They do this by converting the relative heats of the objects, such as a human or a flame, in its view to the background into a gray-scale (How thermal cameras work.).

Visible Light Cameras

Visible light cameras operate using the spectrum of colors that are visible to the human eye. In order to produce the images that people see in the world, visible light cameras capture and process light energy from the surrounding environment and convert it into electrical signals. One of the limitations of visible light cameras, however, is the need of a light source. However, this is not a serious problem as the locations where most boat fires would occur are in places with an abundance of light sources, such as the moon, the sun, and artificial light sources. The fire itself will also provide a source of light.

2.7.2 FireIce Level Sensing

In order to know how much FireIce is left in drone, the team decided to test the effectiveness of a multitude of different sensor types. The different types of sensors that were researched and tested include float sensors, pressure transducers, non-contact sensors, and flowmeters. Through the testing of the sensors, it was found that none of the researched methods would work in the current system. In the end, the team decided to use a simple timer using the total elapsed time operating the pump to estimate the remaining level of the FireIce solution.

Float Sensors

One of the first methods that was looked into were float sensors, specifically magnetic float sensors. There were a few reasons as to why this type of sensor was not implemented in the final design. Firstly, it was determined that the magnetics used within the sensing system could cause interference with the various electrical systems on board the drone, such as navigation and communication. Also, the general orientation of the drone while in flight will cause the FireIce solution in the tank to almost never be level. Due to this, readings from a float sensor would be not be representative of the remaining FireIce solution.

Pressure Transducers

The major problem that was found with using pressure transducers to determine the amount of FireIce solution left in the tank was around the mounting of the transducer. With the tank hanging from the bottom of the drone, additional mounting hardware would have been required in order to create a place to mount the transducer to the bottom of the tank. This would result in extra weight burdened on the drone, which was less than desirable.

Non-Contact Sensors

The use of non-contact sensors was explored greatly, but was deemed impractical due to much of the same reasons as the pressure transducers. The sensors of choice were to be a series of ultrasonic sensors configured in such a manner as to approximately recreate the surface of the solution within the tank via software. Much like the pressure transducers, the main problem with the ultrasonic sensors was where to mount them. The sensors needed to be placed either inside the tank with the solution, which would limit the amount of FireIce we could carry due to the sensors taking up vertical space, or outside of the tank with a piece of the tank cut open. However, with the constant tilting of the orientation of the drone while in flight, holes in the top of the tank could have proven to be troublesome.

Flowmeter

As with all of the other systems tested with the goal of measuring the remaining amount of the FireIce solution, the use of the flowmeters proved to be impossible. Unlike however with the pressure transducers and the non-contact sensors, the flowmeters had easy mounting in line with the tube and the pump. The flowmeters that the team had decided upon gave a digital output in the form of a series of pulses. These pulses could be counted through the use of a software interrupt to determine the flow of the FireIce solution. However, the rate at which the sensor could send the pulses were too fast for the Raspberry Pi to handle given all of the other tasks that it needed to do. Do to the timing limitations of the flowmeters, it was determined by the team that they would not be used.

2.7.3 Communication

MAVLink

As previously mentioned, the last team used the Pixhawk as a controller for the drone. The base sends messages wirelessly to the controller and it responds back, allowing users to navigate the drone. The Pixhawk specifically uses MAVLink to communicate with the base. MAVLink is a lightweight messaging marshalling library that allows for the transmission of telemetry and other types of data to micro air vehicles, and it has gone through extensive testing with PX4, Pixhawk, APM, and AR.Drone platforms (MAVLink micro air vehicle communication protocol.

MAVLink messages are variable in length and are transmitted as strings of encoded bytes as can be seen in Figure 2.4. The payload of the message is prefaced by 6 header bytes: STX, LEN, SEQ, SYS, COMP, and MSG. STX denotes the beginning of a message, LEN contains the max size of the payload, and SEQ is the sequence number of the messages. The fourth byte, SYS, is for the system, the fifth for the component ID, and the sixth is the message header. Following this is the payload which is variable in length. After the payload are two checksum bytes which determine if there has been any error during transmission as per usual for messages that are sent over a network connection (MAVLink micro air vehicle communication protocol.). If these bytes show that there has been a transmission error, the Pixhawk discards the message and waits for the base to send it again.

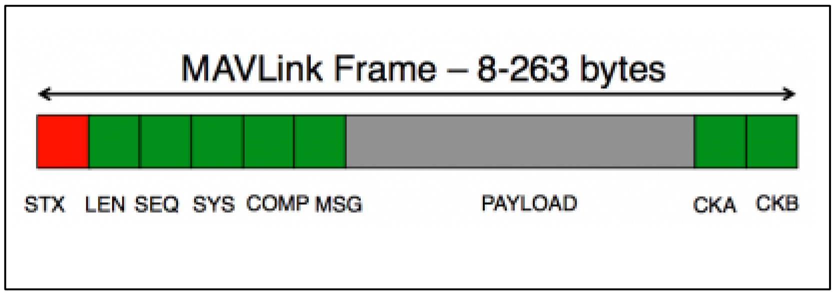


Figure 2.4: MAVLink Message Package Structure (MAVLink micro air vehicle communication protocol.)

Pymavlink

One possible way to decode MAVLink messages from the Pixhawk to the base as well as encode MAVLink messages and send them from the base to the Pixhawk is by using pymavlink. Pymavlink is an open source Python script which reads in MAVLink XML message definitions and generates a module for reading and writing those messages (MAVLink python bindings.2016). There are two ways of running pymavlink, GUI and command line, and it can generate modules in multiple languages: C, C#, Java, JavaScript, Lua, and Python for version 2.7 or later. To select a language, the user must place an XML file containing the definitions for the MAVLink messages of the chosen language in the 'message_definitions' folder and name it common.xml. Once the Python script is run, users can add the resulting file into their program which will allow them to process multiple MAVLink streams at once and parse the messages. Users can then 'pack' messages using various functions depending on the desired message and then serialize it, sending it to the Pixhawk (MAVLink python bindings.2016).

After much consideration, the team decided against using Pymavlink. While Pymavlink would have allowed the team to send MAVLink messages to the Pixhawk in order to control the drone, this would not aid in meeting any of the major requirements of the project.

Bluetooth

Bluetooth, defined by the IEEE 802.15.1 standard, was originally developed to replace cables in personal area networks and is an open specification for short-range wireless voice and data communications. Bluetooth was originally considered for three different applications: wire replacement, ad hoc networking of several different devices in short proximity, and as an access point to the wide area voice and data networks like those provided by cellular networks. Today, Bluetooth is primarily used in the first application, as a wire replacement (Pahlaven & Krishnamurthy, 2013).

The physical connection of the Bluetooth technology uses a FHSS (frequency hopping spread spectrum) modem that gives the technology coverage from a range of 10 meters to about 100 meters. For applications such a computer mouse and laptop, this is fine, but for a drone that needs to across open water to boat fires, possibly miles away, this is insufficient (Pahlaven & Krishnamurthy, 2013).

ZigBee

ZigBee, like Bluetooth is a technology handled by the IEEE 802.15 working group for wireless personal area networks. ZigBee, defined by the 802.15.4 specification, is a suite of protocols capable of networking with nodes that are in range. Where Bluetooth is based off of the IEEE 802.11 FHSS standard using a frequency hopping spread spectrum, ZigBee is based off of the IEEE 802.11 DSSS standard. DSSS, or direct sequence spread spectrum, is a form of data transmission where each information symbol is coded into smaller, narrower pulses also known as chips. These chips are generated as pseudo-noise (PN) sequences. These sequences are generated using a variety of different methods depending on the band of transmission. For the 2.4 GHz band, the PN sequences are generated through the use of 16-ary orthogonal coding with sequences having 32 chips (Pahlaven & Krishnamurthy, 2013).

XBee, the transmission technology used by the previous team, is an application of ZigBee with a much greater range of coverage than that of Bluetooth (XBee® DigiMesh® 2.4.). XBee devices can cover an area up to 28 miles, which is more than acceptable for the system's application.

As such, the team looked further into using it to send messages between the base computer and the Raspberry Pi on the drone. The XBee modules would be connected to the base computer that is running the GUI through USB and the Raspberry Pi through a serial connection. This would allow commands from the GUI to be sent to the drone and for data to be sent from the drone to the base computer to be displayed on the interface.

Furthermore, the XBee models being used have two different modes: Transparent (AT) and Application Programming Interface (API). In AT mode, any data that is sent to the XBee module is immediately sent to the pair XBee module. The transmission is a simple serial TX out and serial RX in communication. This mode only works when you are using two XBees and simple point to point communication. AT mode is the default mode on XBees. API mode uses data packets instead of simple serial transmission. It is preferred for larger networks that involve communication between multiple devices. API uses a master (controller) and slave (router) system that allows a master XBee device to send to and receive from all slave devices. In addition, the API mode has a built in packet delivery confirmation with every transmit package. Both modes will be tested to see which is best.

Thorough tests of the XBee modules' reliability over various distances were performed and found that they transmitted messages without error up until they were approximately 300ft apart. However, the team determined it was best to use XML HTTP requests to send messages to and from the base computer since it would limit the number of communication types being. As such, the XBee modules were not integrated into the system design.

2.7.4 Power Distribution

The previous team used a combination of a 12V buck converter and a 5V step down voltage regulator to create two voltage points for which every electrical component operated from (Peterson et al., 2016). Since this system is already in place, and verified to be working, further research into methods of achieving the voltage drops and components to do so was not done. Therefore, this portion of the PCB design already aligned well with the previous team's design. More information on the design can be found in their report.

2.8 Software Background

2.8.1 Graphical User Interface

The previous team created a Graphical User Interface (GUI) to both control the drone and to monitor its status. However, there was room for improvement within the old system.

According to the previous team, the GUI had latency issues and they were unable to implement certain features as initially intended. For example, in order to spray the FireIce continuously, the program running the GUI had to repeatedly send a signal to the drone. A function was created to check on a flag every 200 milliseconds (Peterson et al., 2016). Also, since the camera on the drone was never fully implemented, the GUI did not support live video, a major requirement of the new system. While the GUI did do its job, it could have benefited from a more intuitive layout. A screenshot of the previous team's user interface can be found below in Figure 2.5.

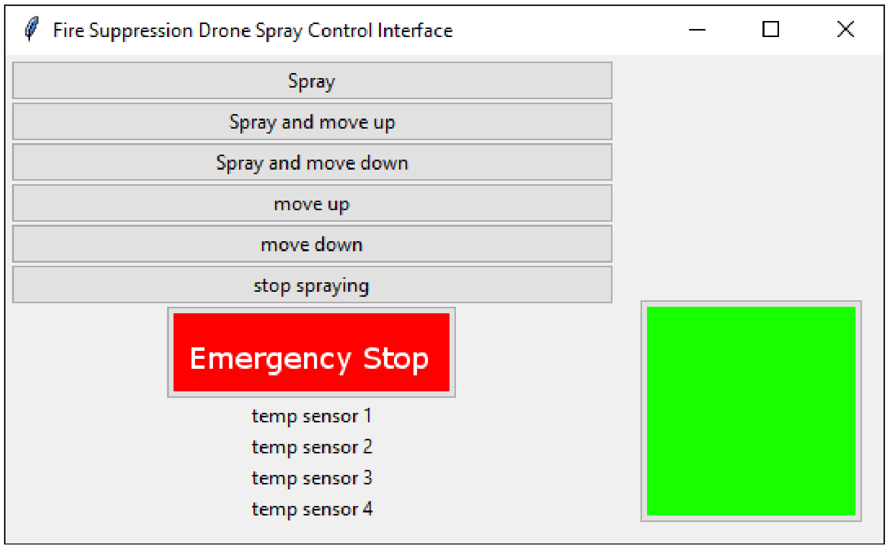


Figure 2.5: Previous GUI design (Peterson et al., 2016)

2.8.2 Image Processing

In order to meet the team's secondary requirement of processing the image, the team first had to determine what type of processing would be most helpful. At first, it was proposed that the drone should be able to detect a fire and autonomously fly to it. This idea was scrapped however, the team decided that having a system to detect a fire would satisfy the requirement of processing the video data and would be a useful addition to the system.

The time required to process the images affects the latency in the system and the team wanted to mitigate this as much as possible. The 2016 De-mining MQP team did some testing of various ways of processing the data from their camera. At first the camera was down to 2 FPS after applying their algorithm to the input. The algorithm that the data was being run through was broken into multiple steps, so they first attempted to run multiple frames through the algorithm at once, assigning each frame to a different part of the algorithm and then switching them around as they finished. This method did not affect the FPS too much, so they attempted to use multiple cores on the RasPi, which has four, to process the frames as can be seen in Figure 2.6. This processing method was effective, increasing the camera back up to 8 FPS and decreasing the latency from 500ms to 200ms (Lockman & Haydon, 2016).

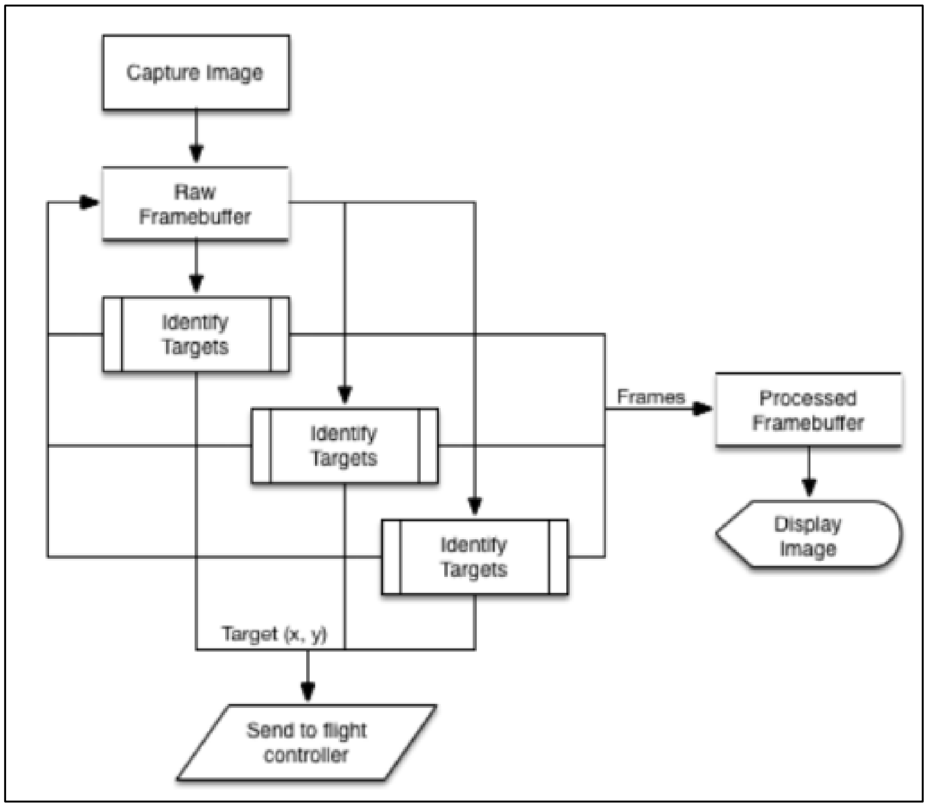


Figure 2.6: De-Mining Image Processing

The 2016 De-mining MQP team also separated their algorithm into a separate script and sent the important data to the main control code through ports. This allowed them to run the algorithm at different speeds than the main code and be blocked if necessary to assure that it would not affect the speed of the main code. As effective as this process was, it also had its drawbacks; it required an asynchronous data-passing method. As such, the previous team implemented a client-server port scheme which controlled the passing of data from the standalone script to the main control code (Lockman & Haydon, 2016).

OpenCV

OpenCV is an open source computer vision and machine learning software library which was originally built to advance machine perception in commercial products and provide a common infrastructure for computer vision applications. OpenCV contains more than 2500 optimized algorithms which can be used to detect and recognize faces, classify human actions in videos, identify objects, etc. Specifically, OpenCV contains functions which could be used to extract fire colors from the drone's camera feed and enable the team to determine the exact location of the fire (About.2016).

Visualization

Receiving the camera data at a usable FPS is necessary when it comes to satisfying the requirement of streaming video to the base computer as outlined in Chapter 1. The 2016 De-mining MQP team tried two methods of viewing the data. The first method was to remote desktop into the RasPi and view it using its operating system. The problem with this was that the WiFi connection was not fast, causing the camera stream to decrease to 0.25 FPS. This was not a high enough rate to be used, so they continued onto their second method which was to send the camera data to a web stream which could be viewed by any connected clients. This was accomplished via a Python script running on the Raspberry Pi. This ended up being the ideal choice as it was quick enough to adequately see the camera stream and be able to control the drone (Lockman & Haydon, 2016).

3 System Architecture

3.1 Mechanical System Architecture

3.1.1 Pre-determined System Parameters

Property	**Value**
Air Density,	1.205 [kg/m^3]
Air Kinematic Viscosity, v	15.11e-6 [m^2/s]
Air Specific Heat, cp	1,005 [J/(kg*K)]
Air Temperature, T	293 [K]
Air Thermal Conductivity, k	0.0257 [W/(m*K)]
Gravity, g	9.81 [m/s^2]

Table 3.1: Global System Properties

Assumption	**Value**
FireIce Density, d	1,200 [kg/m^3]
FireIce Dynamic Viscosity,	1e-3 [Pa-S]

Table 3.2: Global System Assumptions

3.1.2 Spray Pattern

Spray Pattern Distance and Coverage

In order to determine the required spray distance, we had to first find the minimum safe distance of operation from the fire. In finding this value, we only have to worry about heat transfer from radiation due to the fact that convection and conduction will not affect our system as we are neither in contact with a surface and the drone is operating in an open environment. This can be modeled using equation 1 below:

$$r = \sqrt{\frac{Q\lambda}{4\pi q''}}$$

Equation 1: Minimum Safe Distance

In this equation, we solve for “r” which is the minimum safe distance of operation from the source of the fire. The previous team found the minimum distance to equal 14.2 feet with acrylic which has the highest radiative heat transfer rate of all materials on the boats in question. They found other common materials such as wood and fiberglass to have minimum distances of 9.25 feet and 4.7 feet respectively. Our team used the acrylic minimum safe distance, as it was the most limiting case. We also decided that it was beneficial to add a safety factor, "N_d", of 1.5 to add a buffer between the drone and this minimum safe distance as shown in equation 2.

$$r_n = N_d * r$$

Equation 2: Minimum Safe Distance with Safety Factor

This will help protect the drone from getting too close and minimize the chance of an outside influence, such as wind, from pushing the drone into the dangerous zone. Our resulting minimum safe distance with the safety factor was 21.3 feet, or roughly 6.5 meters.

Once we had obtained the minimum safe distance, we then used the distance formula, equation 3, to find the minimum horizontal distance to the fire given a range of our expected operating heights.

$$d = \sqrt{\Delta x^2 + \Delta y^2}$$

Equation 3: Distance Formula

Our operating heights ranged from 2 to 6 meters above the origin of the surface of the hypothetical body of water the drone would fly over. We found the following minimum horizontal distances to the fire in Table 3.3:

Height (m)	**Minimum Horizontal Distance to Fire (m)**
2	6.18
3	5.77
4	5.12
5	4.15
6	2.5

Table 3.3: Minimum Horizontal Distances to Fire based on Drone Operating Height

Based upon Table 3.1, we know that our minimum horizontal distance to the fire is limited by the 6.18 meters at a height of 2 meters. Using this case, we were able to model the FireIce as a projectile motion problem to solve for the exit velocity required at the nozzle. Using the projectile motion equation, equation 4, the mass flow rate relationship, equation 5, and volumetric flow rate relationship, equation 6, we were able to find the needed outlet conditions from the nozzle to satisfy our worst case scenario for a minimum safe distance to the fire.

$$Y_f = x_f \tan \theta - \frac{g x_f^2}{2 v_0^2 \cos^2 \theta}$$

Equation 4: Projectile Motion

$$Q = \rho * v * A$$

Equation 5: Mass Flow Rate Relationship

$$V = v * A$$

Equation 6: Volumetric Flow Rate Relationship

When performing these calculations, we assumed our degrees of freedom of the nozzle could rotate 90 degrees, 45 degrees upwards, and 45 degrees downward. For our analysis, we only considered upwards pitch because downward pitch would only be needed to adjust the spray and wouldn't require extra pressure. The projectile motion equation was assessed at various angles (0, 15, 30 and 45 degrees). The solid stream and fanned nozzle candidates had an option for an outlet diameter of 0.04 inches, which we assumed for this analysis. This made our cross sectional area 0.810732 mm^2. Using this information, we then found the required outlet flow rates and velocity. The results are shown below in Table 3.4:

Angle	Required Outlet Velocity (m/s)	Required Outlet Mass Flow Rate (kg/s)	Required Outlet Volumetric Flow Rate (Liters/s)
45	6.77	6.59e-3	5.49e-3
30	6.70	6.52e-3	5.43e-3
15	7.41	7.21e-3	6.01e-3
0	9.68	9.42e-3	7.85e-3

Table 3.4: Calculated Required Outlet Velocities and Flow Rates

From this analysis we found that after pressure and flow losses leading up to the nozzle that our maximum required outlet mass flow rate is 9.42e-3 kilograms per second and a volumetric flow rate of 7.85e-3 liters per second, which occurs at a nozzle angle of 0 degrees.

Spray Design

In order to determine the optimal atomization of FireIce to adequately contain a boat fire, we had to calculate the optimal droplet size. Our team decided to calculate for the worst case scenario which we determined to be a gasoline induced boat fire. In calculating this, numerous properties were needed, shown in Table 3.5, and assumptions, shown in Table 3.6.

Property	**Value**
Gasoline Density, gas	719.7 [kg/m^3]
Gasoline Specific Heat, cgas	2,200 [J/(kg*K)]
Gasoline Thermal Conductivity, kgas	0.1 e-3 [W/(m*K)]
Gasoline Kinematic Viscosity,	8.8 e-7 [m^2/s]
FireIce Initial Velocity, V0	9.68 [m/s]
Drone Minimum Height, hmin	2 [m]
Drone Height, hmax	{2<=h<=6} [m]

Table 3.5: Optimal Droplet Diameter Properties

Assumption	**Value**
Gasoline Temperature, Tgas	1300 [K]
Fire Origin,h0	0 [m]
Droplet Surface Temperature,Ts	294 [K]
Fire Convective Heat Release Rate, Qc'	4.5 [kW] *Average Reported Methane HRR

Table 3.6: Optimal Droplet Diameter Assumptions

There were two main cases to consider when solving this: an evaporating case and a non-evaporating case. The evaporating case of eliminating the fire is to use a finer mist to cool the flame itself to eliminate and suppress the fire. The non-evaporating case is where the FireIce would ideally penetrate the fire plume and cool the fuel or origin of the flame. The evaporating case is found using equation 7.

$$d_{opt} = \sqrt{36v * \frac{\rho_{gas}}{\rho_d} * \frac{\frac{-V_0 h_0}{2} + \frac{k_{ev} V_0}{4} + \int_{h_{max}}^{0} U(h)dh}{0.5V_0^2 - \frac{g h_0}{2}}}$$

Equation 7: Optimal Droplet Diameter for Evaporating Situation

U(h) in equation 7 is the velocity of the fire's plume that is generated. This value is a function of the plume height which is found using equation 8.

$$U(h) = 3.4\left(\frac{g}{c_{gas} T_\infty \rho_\infty}\right)^{\frac{1}{3}} Q_c'^{\frac{1}{3}}(h - h_0)^{-1/3}$$

Equation 8: Fire Plume Velocity

Using equation 8 we found the average plume velocity to be a rate of 4.94 meters per second. Our team also created a table of the plume velocities at the various distances away from the origin as shown below in Table 3.7.

Height (m)	Fire Plume Velocity (m/s)
0.001	13.07
0.01	6.07
0.1	2.82
1	1.31
2	1.04
3	0.91
4	0.823
5	0.76
6	0.72

Table 3.7: Fire Plume Velocities based on Height from origin

In order to solve equation 7 we also had to determine the evaporation constant, kev, using equation 9. This gives us a coefficient representative of the rate of evaporation based on gasoline properties and our environmental properties.

$$k_{ev} = \left(\frac{8k_{gas}}{\rho_{gas}c_{gas}}\right) * \ln\left(1 + \frac{c_{gas}}{L}\left(T_{gas} - T_s\right)\right)$$

Equation 9: Evaporation Constant

Our team found kevto equal 6.37e-9. Using this constant, our fire plume values, and the properties previously mentioned, we were able to discover that the optimum droplet diameter for the evaporating case to be 1.58 mm.
Next, we proceeded to assess the optimal droplet diameter for the non-evaporating situation using equation 10.

$$d_{opt} = \sqrt{36v * \frac{\rho_{gas}}{\rho_d} * \frac{\int_{h_{max}}^{h_{min}} U(h)dh - \frac{|V_0|}{2}(h_{max} - h_{min})}{g(h_{min} - h_{max}) - 0.5V_0^2}}$$

Equation 10: Optimal Droplet Diameter for Non-Evaporating Situation

From this equation, our team found that the non-evaporating case had an optimal droplet diameter of 4.32 mm. This was the droplet diameter size that our team strived to achieve in designing our spraying system.

To determine our actual droplet diameter mathematically, our team used Weber's Equation, equation 11. In order to solve for this, our team had to find the Ohnesorge number, the Weber number, and the Reynolds number using equations 12, 13, and 14 respectively. It is important to note that the "d" in the equation is the nozzle outlet diameter.

$$\lambda_{opt} = 4.44d(1 + 3Z)^{0.5}$$

Equation 11: Weber's Equation for Droplet Diameter

$$Z = \frac{\mu_L}{(\rho_L \sigma \mathrm{d})^{0.5}} = \frac{\sqrt{W_e}}{R_e}$$

Equation 12: Ohnesorge Number

$$W_e = \frac{\rho v^2 d}{\gamma}$$

Equation 13: Weber Number

$$R_e = \frac{\rho v d}{\mu}$$

Equation 14: Reynolds Number

After solving these equations, we found our droplet diameter of the FireIce to be 4.53 mm, which is very comparable to the 4.32 mm optimal droplet diameter we had calculated previously. This confirmed our team's decision to pursue the solid stream and fanned nozzles with the 0.04 inch outlet diameter.

Fluidic Oscillator Design

In designing the fluidic oscillator, a test subject was developed and simulated in SolidWorks, as shown above in Figure 3.1. This simulation showed promise as it was able to create lift between the inlet stream and the outlet. This causes the spray pattern to oscillate back and forth as it exits the nozzle.

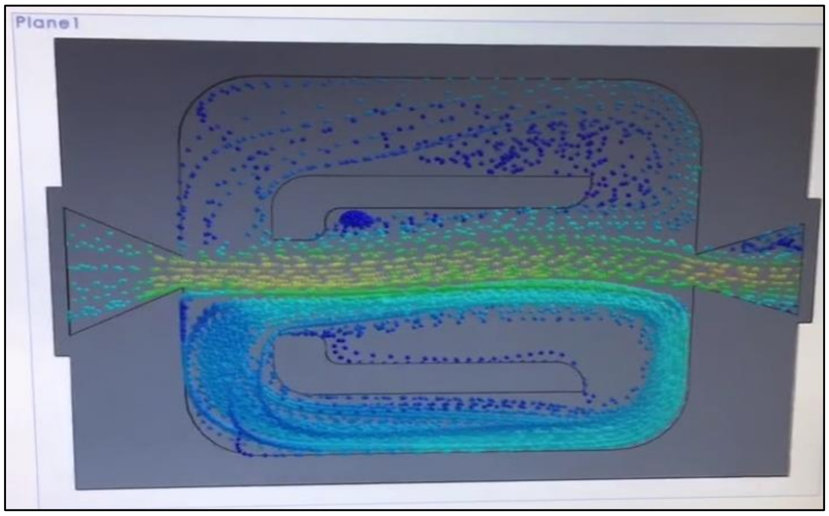


Figure 3.1: Fluidic Oscillator Simulation

While this simulation was able to prove that this design is possible, there were many issues with it. In order to create this effect, our nozzle would require a diffuser, as seen on the right of the system in Figure 3.1. Based off of our previous calculations, we would need a nozzle outlet of 0.04 inches in diameter to generate the adequate velocity. The fluidic oscillator design requires a wide outlet nozzle to perform as expected, and in the simulation done, it required a minimum of 0.4 inches to oscillate even a slight amount. While it is possible to increase the pressure to the system to allow for an optimal outlet diameter to make this design possible, our drone is limited to a 24 pound payload, and as will be later calculated, our team found that under the weight restrictions of the drone, we could get no more than 160 psi out of the system. With such a large orifice at the outlet we later calculated a need for 275 psi, dispelling any chance for this nozzle solution to work.

3.1.3 System Degrees of Freedom

The addition of the single degree of freedom at the nozzle is beneficial to increasing the range on the spray as well as for adjusting it to better target the fire. Our team modeled the FireIce in the projectile motion equation, equation 4, to create a table of distance based on the operating height and spray angle as shown in Table 3.8.

Based on this data we were able to quantify our expected spray distances and found that we get the maximum distance when oriented at 30 degrees. At our operating height of 6 meters and angle of 30 degrees, we would achieve a spray distance of 14.29 meters which is a considerably long range. This data also confirmed to us that a 45 degree range in the positive and negative is optimal for adjusting the spray.

Height	Angle	Calculated Distance (m)
2	0	6.18
2	15	8.82
2	30	10.90
2	45	11.25
3	0	7.57
3	15	10.08
3	30	11.89
3	45	11.95
4	0	8.74
4	15	11.16
4	30	12.76
4	45	12.59
5	0	9.77
5	15	12.13
5	30	13.56
5	45	13.18
6	0	10.71
6	15	13.00
6	30	14.29
6	45	13.73

Table 3.8: Spray Distances based on Operating Height and Spray Angle

3.1.4 Pumping System

One of the most critical components of the mechanical aspects of this project is the pumping system. In order to determine the pumping systems requirements a series of calculations were needed. Firstly, our team had to find the pressure required at the inlet of the nozzle. The nozzles that we found from the spray pattern analysis had an inlet diameter of 0.25 inches. Using this data, we used the volumetric flow rate equation, equation 6, and created a volumetric flow rate relationship as shown in equation 15. This relationship shows that volumetric flow is constant throughout a system and we used that characteristic to find the inlet velocity at the nozzle.

$$A_i v_i = A_0 v_0$$

Equation 15: Volumetric Flow Rate Relationship

We found the inlet velocity to the nozzle had to be 0.248 meters per second. Based upon the nozzle ratings, and a required 0.124 gallon per minute rate of flow, we found that we needed a 138.7 psi outlet pressure. Knowing this value, our team used Bernoulli's Equation, equation 16, to find the required inlet pressure to the nozzle.

$$\frac{P_2 - P_1}{\rho} = -\frac{v_2^2 - v_1^2}{2}$$

Equation 16: Bernoulli's Equation

This equation allowed us to find that we needed an inlet pressure to the nozzle of 146.85 psi. The last step was to determine the pressure loss over our tubing and piping connections from the pump to the nozzle. Our tubing solution consisted of two main parts, one made up of 0.45 meters of plastic tubing and 0.1 meters of steel pipe, for a total of 0.55 meters. By using the pressure loss equation, equation 17, the f coefficient equation, equation 18, and Reynolds number we were able to determine the pump pressure required for this system to function.

$$\Delta P = f(L/D) * \frac{\rho * V^2}{2}$$

Equation 17: Pressure Loss

$$f = 64/R_e$$

Equation 18: f coefficient

The f coefficient came out to be 0.0287 and we found our pressure loss to be 58.25 Pascals, or a 0.00845 psi difference, which is expectedly low due to the short range of operation. This results in our system requiring 146.86 psi from the pumping system. Knowing that pumps do not provide perfect efficiency, we factored in a 90% efficiency rating to find the max pressure our pump needed to be rated for. This gave us an ideal max pressure rating of 163 psi.

FireIce Containment System

The design of the FireIce containment system was determined based upon weight restrictions from the max payload. We decided to design the tank to contain as much FireIce as possible with the remaining amount of weight capacity remaining. The primary limiting factor in the tank design was the placement of mounting fixtures for the mounting plate's clamps to the drone. Avoiding the need for standoffs and keeping the tank within the width parameter of the mounting plate itself, the maximum surface area it could have was 1.89e4 mm^2. Having the width and length of the tank parameters fixed, the tank was designed primarily around the height dimension. The final volume was 1.5e6 mm^3, or 1.5 Liters.

Based upon our volume and volumetric flow rate through the system, we calculated the time that it would take to completely empty the tank of FireIce from the system. With a flow rate of 7.85e-3 liters per second and a volume of 1.5 liters, it would take approximately 3 minutes and 11 seconds to discharge all of the FireIce from the system.

3.2 Electrical System Architecture

The electrical system architecture is composed of two subsystems. These subsystems are comprised of the camera system and the power distribution system. Section 3.2.1 will briefly talk about the specific requirements needed to be met for the camera system and section 3.2.2 will discuss the specific voltage requirements for each of the different components in the electric system.

3.2.1 Camera System

The camera system used to meet the visualization requirements for the drone needed to have certain characteristics. The first of these requirements is that the camera needed to be able to stream a video. The second requirement is that the camera system needed to take the video with a resolution that will allow digital image processing techniques to be employed. The final requirement of the camera system is that is needed to be easily configured to work alongside the rest of the system.

3.2.2 Power Distribution System

The power distribution system for the system includes a 12V source as well as a 5V source. The combination of these sources powers the entirety of the board. Table 3.9 below describes the voltage requirements for each of the components on the board. The currents needed for each of the various electrical and mechanical components varied and is also shown in the table below.

The pump needed a 12V source and 8.33A to run at its peak. These requirements are not hard requirements and can be under the specified amounts, but for optimal performance, they should be met. The H-bridge needed to have both a 5V source for the logic operations performed on the chip and a 12V source to power the DC motor connected to the H-bridge. The current needed for the H-bridge is reliant on the current requirements of the DC motor. The current requirement for the DC motor was significantly less than what was needed for the pump and was not considered during the design of the power distribution system. The ADC also needed the 5V source to use as a reference in the quantization for the analog signals coming from the temperature sensors. The current requirement of the ADC, was determined to be quite low at 500μA. Both of the different types of sensors used needed a 5V source as well. The current requirement for the temperature sensors and the flowmeters was deemed to be met by the 5V source that the previous team used, the D24V50F5 5V, 5A step down voltage regulator.

Device Name	Voltage Requirement	Current Requirement
Pump	12V	8.33A
H-Bridge	12V & 5V	<8.33A
ADC	5V	500μA
Temperature Sensors	5V	<5A
Flowmeters	5V	<5A

Table 3.9: Voltage and current requirements for the various components in the electrical system

3.3 Software Architecture

In order to properly control the system on the drone and monitor both the drone and system's status, the team needed to develop two main pieces of software: the GUI and the Data and Control Server.

3.3.1 Graphical User Interface

The GUI needed to be designed so that it was capable of displaying and transmitting a variety of data. The main features required for sending commands to the drone were five buttons: Start Spray, Stop Spray, Rotate Up, Rotate Down, and Fire Detection. The first four buttons needed to turn the spraying on and off and control the vertical angle of the spray. The last button would enable and disable the fire detection image processing on the camera stream.

The GUI would also display data from the camera, and temperature sensors, as well as the estimated FireIce level remaining in the tank. The camera stream would be 640 pixels by 480 pixels so that users can easily determine what is being displayed. The team wanted the temperature sensors to be displayed in a way which would be aesthetically appealing while also allowing it to easily warn the user if the drone is getting too hot. Lastly, the FireIce level would be displayed so that it the user can visibly see the tank level decreasing.

3.3.2 Control Server

The server's main purpose is to watch for messages from the GUI, control the pump, rotate the nozzle, process the image from the camera, and respond to the GUI with the temperatures from all of the temperature sensors. As such, the team needed to design the server so that it would be able to run all of these tasks efficiently, not causing any delays in its response to the GUI. The image processing was a main concern for causing a lag in the camera stream, so the team took extra care in making sure the image processing would allow for a high enough frame rate, 15 FPS.

3.3.3 Communication

Having a reliable and speedy connection was a high priority for the team. Without that, there could be an added delay to the server's response to the GUI which, as already discussed above, is to be avoided. Additionally, the communication would ideally be easy to understand, implement, and add on-to incase the team decided it would like more communication between the base computer and Raspberry Pi.

4 Final Design

4.1 Mechanical

4.1.1 Spray Pattern

The nozzle is responsible for pressuring the FireIce so that it reaches the target. Based upon the results from the decision matrix, the most viable candidates for nozzles and relative spray patterns were a solid stream, a flat-fanned nozzle spray, and a fluidic oscillator. Based upon our calculations, a fluidic oscillator was deemed impossible due to our pump limitations. This had left our team with the decision between a solid stream and flat-fanned nozzle spray.

The solid stream atomizes to a droplet diameter very near the optimal droplet diameter and has the highest probability of reaching the target without too much worry of the stream being carried off by wind variances. The flat fanned 15-degree nozzle allows for a richer longitudinal coverage than the solid stream would provide and it would be able to cover a more ideal surface area. The tradeoff is that it has a greater chance of be carried off by the wind as the stream thins out.

After testing both the solid stream and the fanned nozzle solutions, it was quickly determined that the solid stream nozzle was the ideal candidate. Initial testing proved that the wind fluctuations were too much for the fanned nozzle to handle and the spray was carried away only 5 to 7 feet from the outlet. The solid stream, however, was not easily deterred by the wind forces and operated as expected. The resulting nozzle that our team chose was a Type 416 stainless steel solid stream nozzle rated for 1.5 gpm at 1000 psi having an inlet diameter of 0.25 inches and an outlet diameter of 0.04 inches.

4.1.2 Degrees of Freedom

The final design implemented the single degree of freedom system, located at a joint attached to the nozzle assembly. This added degree of freedom allows the nozzle angle to be adjusted approximately 60 degrees. The reasoning behind adding this feature is that it allows for the adjustment in projectile motion of the FireIce solution as it exits the nozzle. In the end, the team decided to go with the DC Motor because of its low weight, robustness, and it was able to be controlled relatively well when used with a potentiometer. The motor also operates at 80 rpm at 12V and 40 rpm at 6V, which is slow and practical for our application which will primarily consist of small and infrequent rotations.

This motor was then incorporated into our master assembly file for the drone, and subsequent mounting and adaptor components were integrated to couple the motor with the nozzle and the mounting plate itself. The output shaft of the motor has a hub to hub connection. The second hub connection is custom made with a 3D printer and attaches to an identical component to securely fasten the nozzle. The DoF subsystem that integrated the nozzle adaptor piece can be seen in Figure 4.1 below

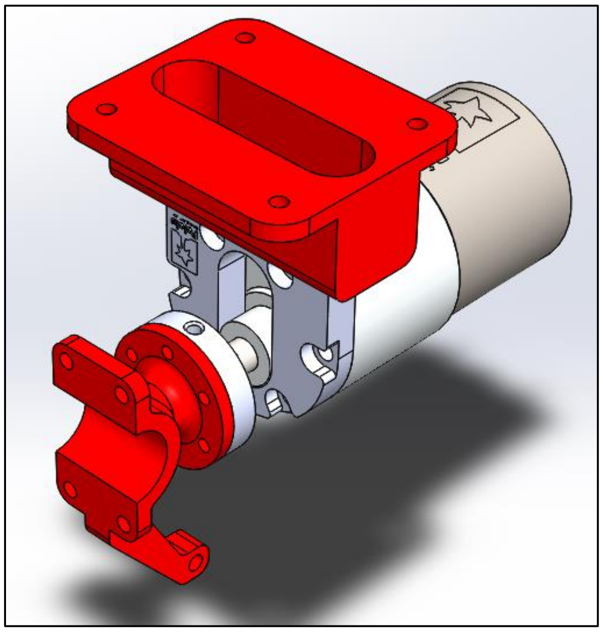


Figure 4.1: DoF Subsystem

4.1.3 Pumping System

Pump

In order to get the FireIce from the tank to the nozzle and generate the flow required, a pump was needed. Based upon the results of the team's decision matrix, as shown in Table 2.6, the top 3 choices for viable pump candidate types were the centrifugal pump, the diaphragm pump, and an internal gear pump. All of these pumps are powered by an integrated motor, which requires a 12V input based upon the standard operation voltage required for the performance range needed for the system's operation.

Originally, the team had concluded on a gear pump sold by McMaster which had an exceptional range of output flow rate, but this pump was deemed impossible for our application due to its size and weight. In the end, the team decided upon a diaphragm pump, the sole pump discovered that was better than the previous MQP pump and within our weight requirements, with the following specifications: 12V DC, 100W, a max of 160 psi, and a max of 8 L/m.

Tubing

The tubing selection is critical for the performance of the DoF, as well as the implementation of the connections between major system components. The nozzle DoF subassembly, as well as future adaptation to the nozzle that requires a controlled flow, needs a rigid and relatively coaxial approach to the nozzle input. To accomplish this, the team decided to implement a blend of rigid piping and flexible tubing for the connection from the nozzle input to the pump output. The connection from the output of the tank to the input of the pump was to be very flexible so the tubing was used.

For the selection of the tubing material, the focus was on flexibility, opacity, maximum operation pressure, maximum operation temperature, and its inner and outer diameter which were dimensions based upon other components. In the end, a stainless steel pipe was chosen for the connection to the nozzle, which is rigid, straight, and has very good corrosion resistance. Using a barbed adaptor, this is connected to the flexible tubing that in turn connects to the outlet of the pump. The flexible tubing that was chosen was high-pressure PVC tubing that had a soft hardness rating, high flexibility, and could withstand a maximum pressure of 220 psi, which is well above the maximum pressure of 160 psi that can be generated by the diaphragm pump. To prevent sagging or twisting of the flexible tubing, guides in the form of eyelets are going to be used to control the path of the flexible tubing as it approaches the nozzle. This same tubing was implemented between the pump inlet and the tank outlet.

4.1.4 FireIce Containment System

The tank is an important aspect of the entire system, as it has the responsibility of holding the FireIce solution, of maximizing the volume that can be held without getting in the way of other system components, and of being easily accessible.

The tank was designed in such a way that the FireIce is guided to the tank exit by a rounded base that directs the flow to the output. In addition, the output is angled downwards, while the tubing that channels to the pump is directly upwards, creating a dry system. Suction from the pump draws the FireIce out of the tank. The final design for the tank design is shown in Figure 4.2 below.

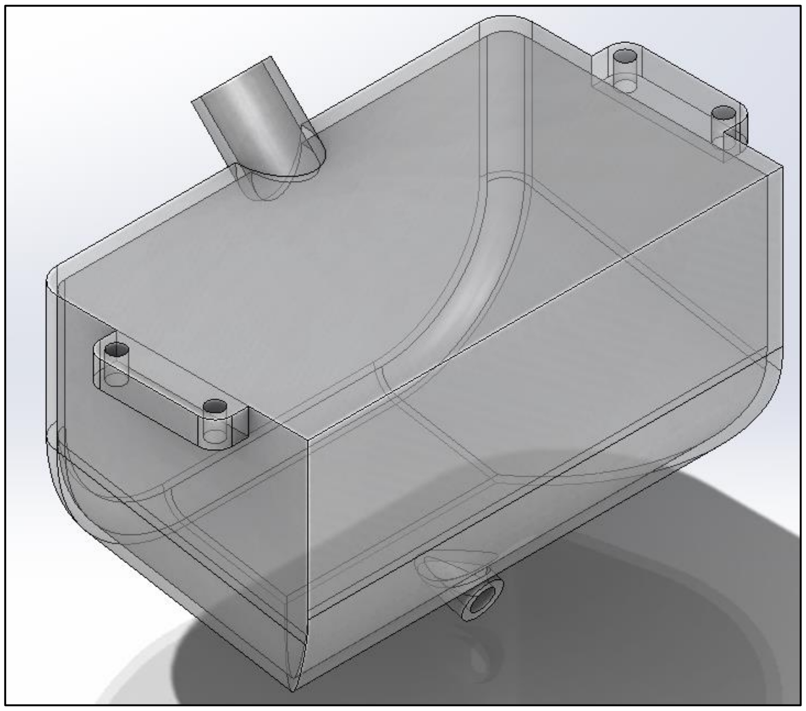


Figure 4.2: Tank Design

The physical tank was created using a fiberglass mold. To do this, a casting structure was made out of cut up MDF blocks and milled down using a CNC to the appropriate dimensions to match the features of the tank. For mounting purposes, the fiberglass was drilled out on the flanges to correspond with the matching holes on the mounting plate. The milling process can be seen in Figure 4.3.

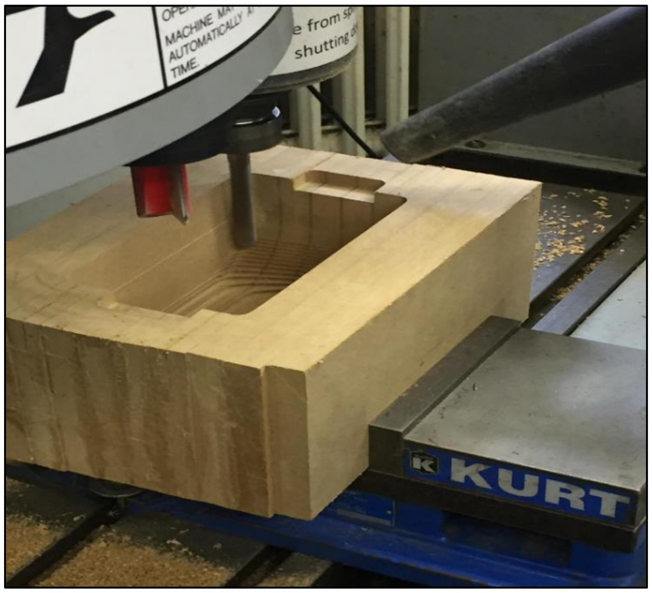


Figure 4.3: Milling the Tank Mold

Once the milling process was completed, 3 coats of polyurethane was applied to the mold and then an anti-sticking agent was applied as well. Once that was accomplished, 5 layers of fiberglass was plastered to the mold and coated with a specialized gluing agent. The result of this step is shown in Figure 4.4.

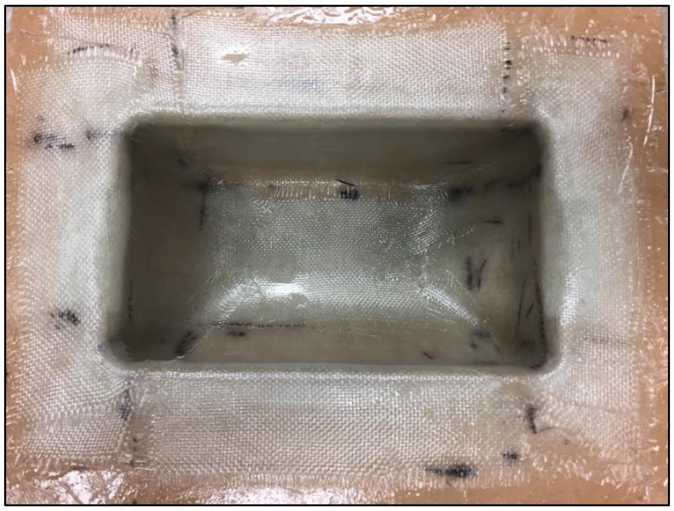


Figure 4.4: Fiberglassing the Mold

Once the fiberglass was dry, the team removed it from the MDF mold and drilled out its mounting holes, the inlet and outlet holes, and attached the tubing adaptors to it. A critical aspect was the application of caulking, or sealant, around the threaded PVC to hold in FireIce and prevent leakage.

4.1.5 Overall Mechanical Design

The integration of the entire system began at the design level in SolidWorks. To mount the mechanical system to the drone, custom shaft collars were created to fit the mounting plate to the rails of the drone. The mounting plate itself was used to attach the separate components such as the tank, pump, and nozzle system. Custom mounts were made for the integration of the DC motor and nozzle mounts. The entire mechanical system design is shown below in Figure 4.5. This design does not include the flexible tubing for simplicity.

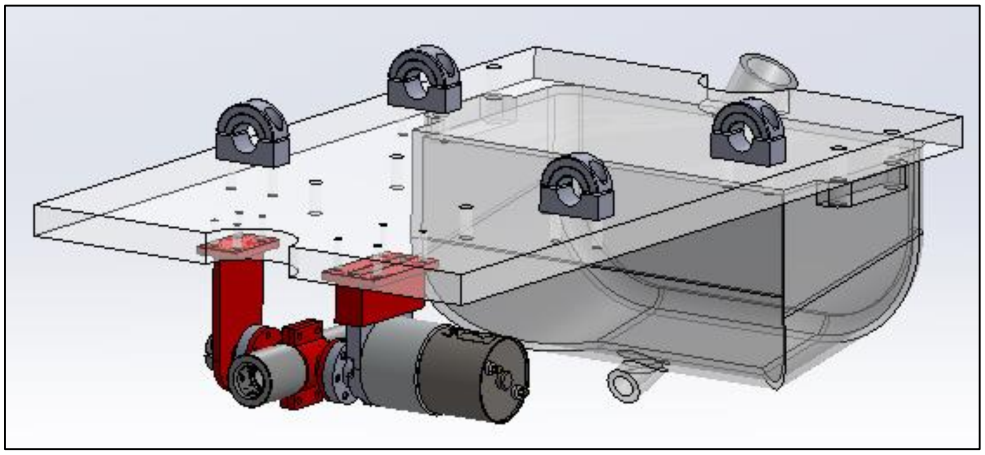


Figure 4.5: Final Mechanical System

Weight Distribution

The weight was one of the most important things to consider when the drone was being designed. The maximum weight the drone can carry is 24.251 lbs. By using SolidWorks, we were able to also ensure that the center of gravity of the system was as centrally located as possible because the drone would experience significant drift if we neglected this factor. The individual component weights can be seen below in Table 4.1.

Item	**Weight**
S1000 Drone	9.26 lbs.
Battery	5.3 lbs.
Tank (Filled with FireIce)	3.8 lbs.
Mounting Plate	1.78 lbs.
Pump	2.75 lbs.
DC Motor	0.518 lbs.
Motor Mount	0.031 lbs.
Motor Hub (x2)	0.029 lbs.
Nozzle	0.05 lbs.
Piping System	0.06 lbs.
Fasteners	~.05 lbs.
Nozzle Pivot	0.03 lbs.
TOTAL	23.687 lbs.

Table 4.1: Component Weights

4.1.6 Mounting Plate

The mounting plate is an important component as it is the component that connects the tank, pump, and DoF system to the drone platform. It features semicircle cuts in the front in the back to allow for the nozzle to rotate and to provide space for the inlet of the tank, respectively. The plate is 0.3125 inches thick, a relatively decent thickness to minimize acute deflection and bending moment from the mounted components. In addition, it is made of strengthened acrylic which has a yield strength of 18 ksi and a melting point of roughly 320 degrees Fahrenheit, making it a decent candidate for mounting and for the potential high temperatures it could be exposed to. The final mounting plate design can be seen below in Figure 4.6.

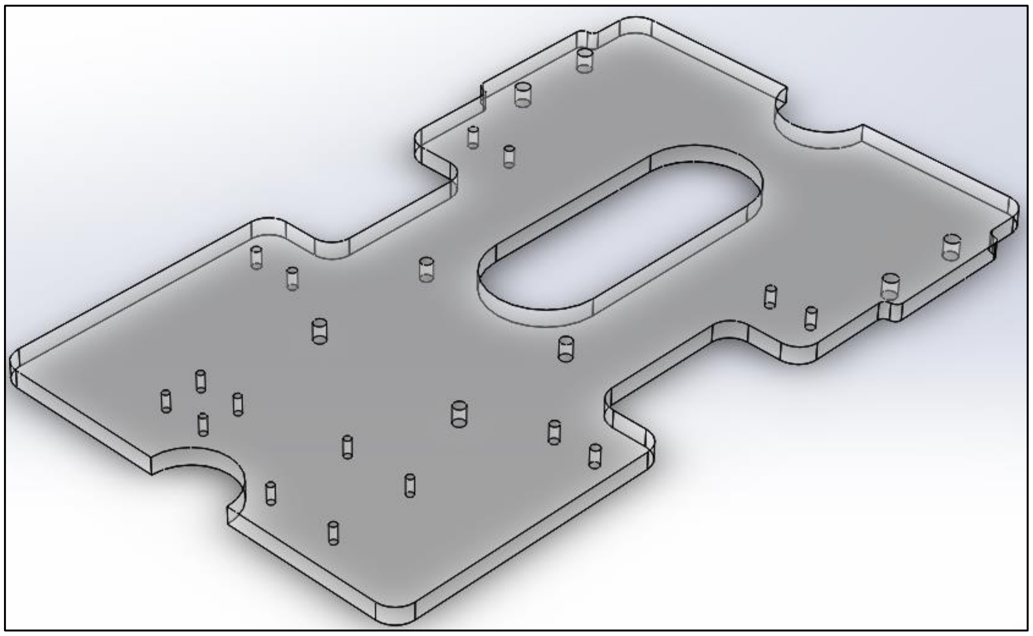


Figure 4.6: Mounting Plate

The team also analyzed the mounting plate to ensure that it would not experience any failures. A Solidworks simulation was run to find the von Mises stress and compare it to our yield strength, as shown below in Figure 4.7. As the simulation shows, the mounting plate is well designed and is well within our parts yield strength, even at the most likely features for failure on the part which are the bolted components between the mounting plate and the drone itself as the entire load is supported at these points.

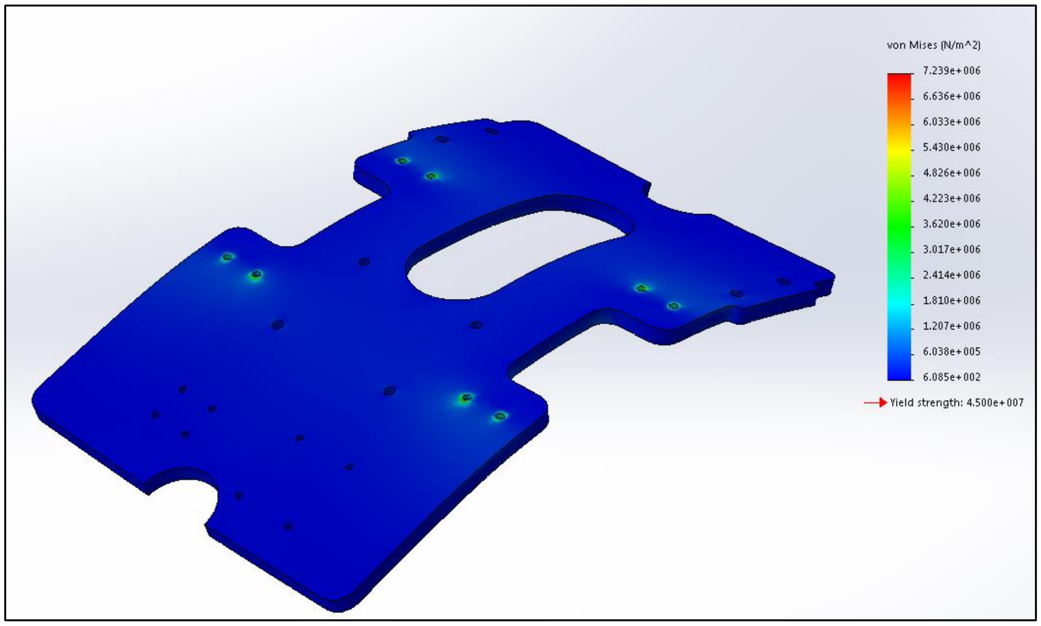


Figure 4.7: Stress Concentrations on the Mounting Plate

4.2 Electrical Design

4.2.1 Schematic Design

Figure 4.8 below shows the H-Bridge driver circuit, used to drive the two DC electric motors connected to the GPIO pin 13 and the GPIO pin 16. The main part of the circuit is the SN754410, a quadruple half-H driver. The GPIO pins 19 and 21 connect to the driver's enable pins, enabling the control of the two DC electric motors connected via pins 3/6 and pins 11/14. The GPIO pins 13 and 16 are connected to the two sets of control signals corresponding to the two DC electric motors. In order to prevent contradictory control signals to be applied to the pins 3/6 and 11/14, two 74HC1G04DCK inverter gates were set between the GPIO pins and the chip.

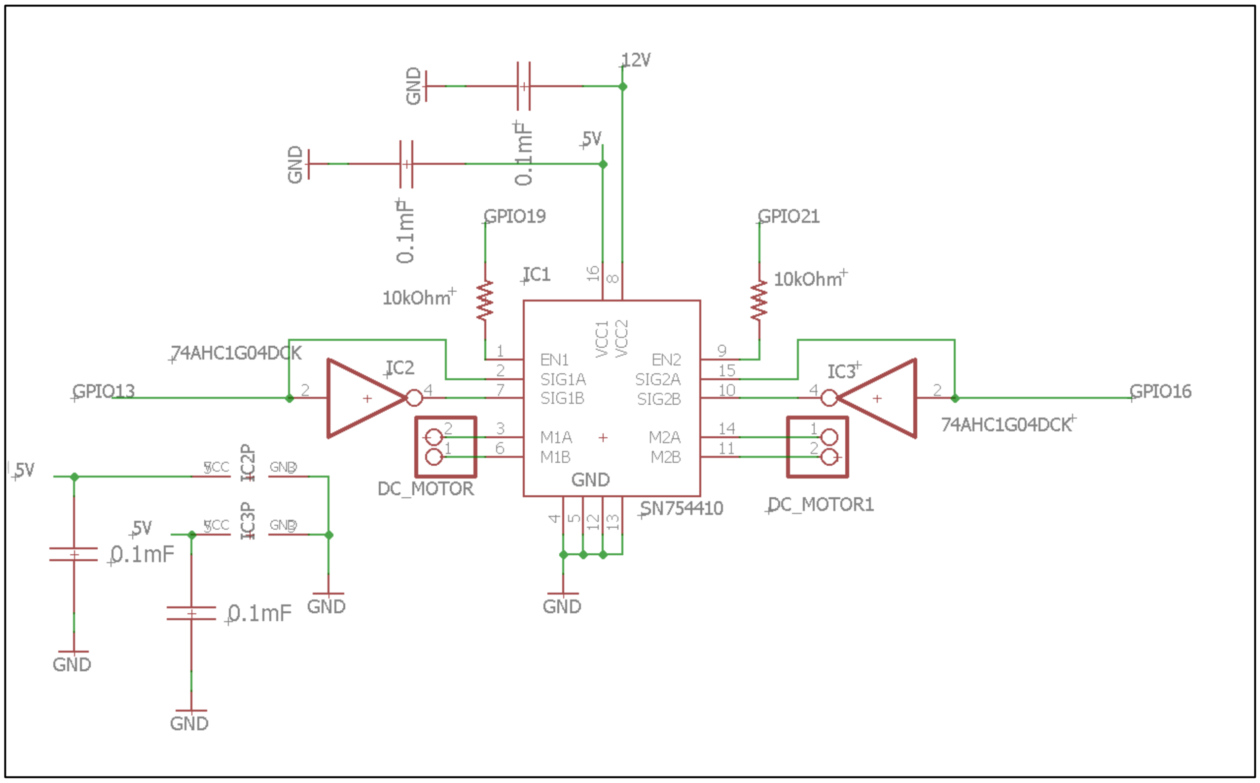


Figure 4.8: H-Bridge Driver Circuit

The 12V buck converter circuit, shown below in Figure 4.9, is composed of two main components. The first of these components is the LM22678, a 12V buck converter. The primary purpose of this chip is to take the 22V battery and to create a 12V source used to the pump.

The second component of the circuit is the control of the pump. Controlled by the GPIO pin 20, the IRF520 Power MOSFET powers the pump with the 12V generated by the LM22678.

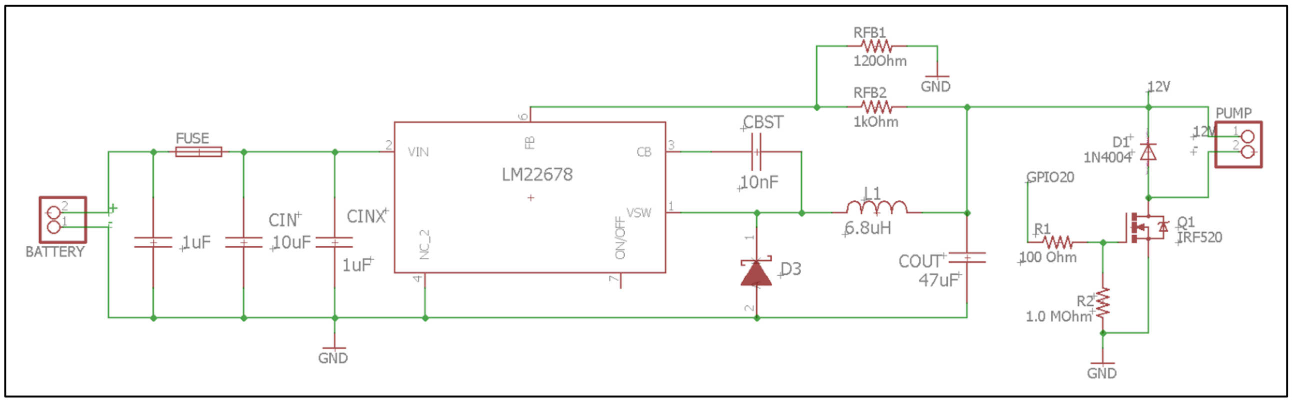


Figure 4.9: 12V Buck Converter Circuit

Shown below in Figure 4.10, the analog-to-digital converter (ADC) circuit consists of a MCP3008-I/SL eight channel ADC connected to four temperature sensors through channels zero through three. Channels four through seven are wired for general use. The pinouts for each of the channels are configured to support a three-wire format: a ground wire, a power wire, and a signal wire.

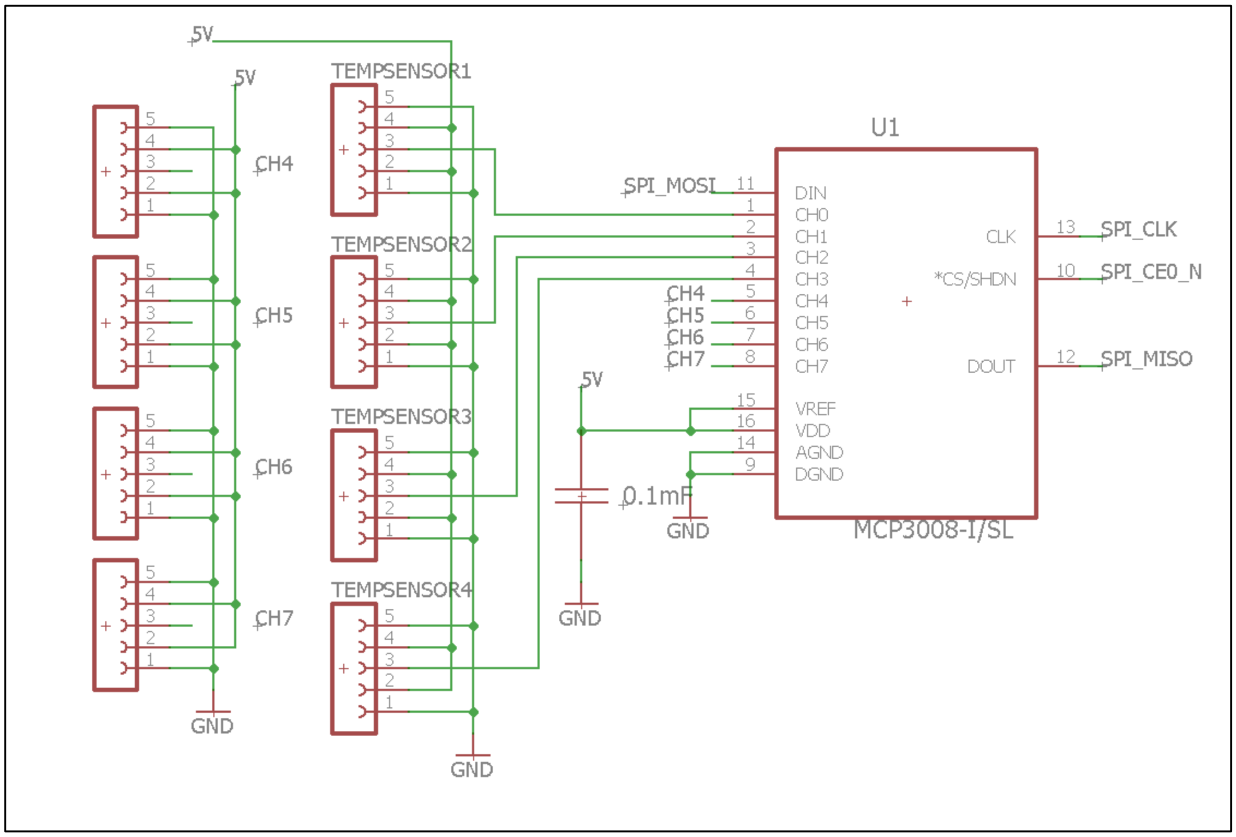


Figure 4.10: Analog-to-Digital Converter Circuit

The GPIO header circuit, shown below in Figure 4.11, consists of two parts. The first part is shown on the left and is 2x20 male pin header for use with the ribb-o-tron 5000 from the Raspberry Pi. The second part of the circuit, shown on the right, consists of pinouts for the different ports from the rib-o-tron 5000.

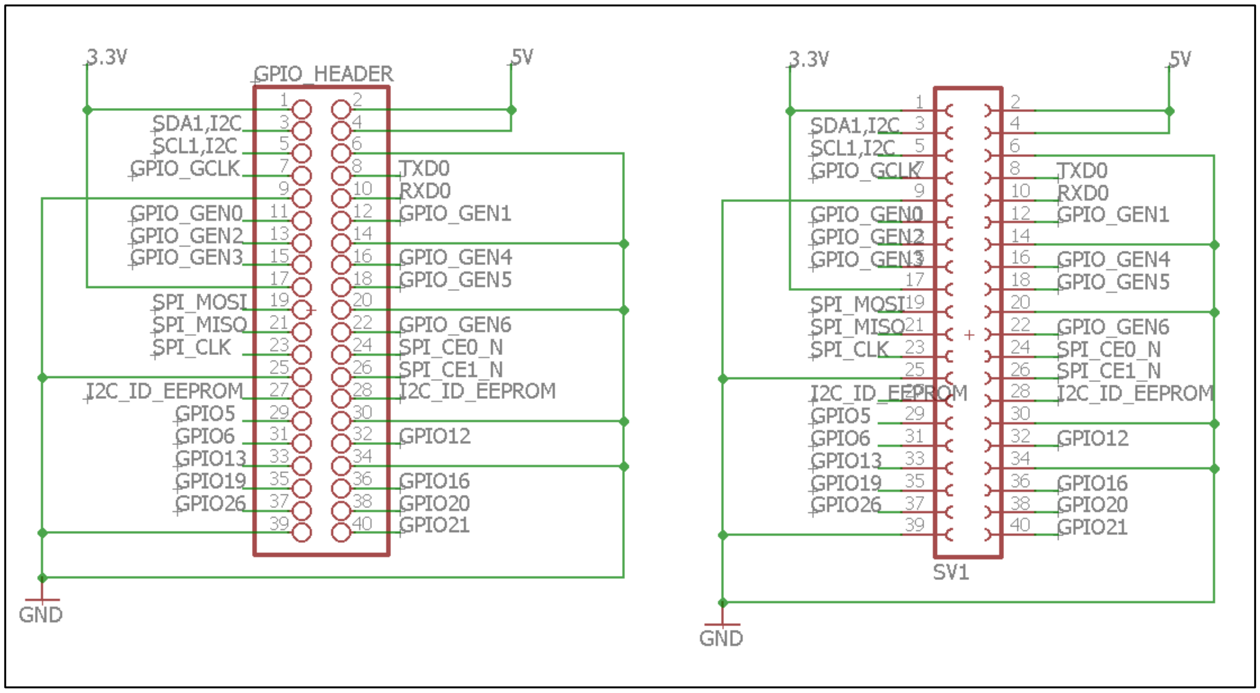


Figure 4.11: GPIO Header Circuit

The last portion of the schematic, as shown below in Figure 4.12, shows two connectors for two digital output flow meters connected to GPIO pins 5 and 6. In addition, there is a pinout for each of the different voltage levels used across the board: 12V, 5V, and 3.3V. Lastly, there is a connection for the 5V step down voltage regulator, D24V50F5. The D24V50F5 takes in the 12V source and generates the 5V source.

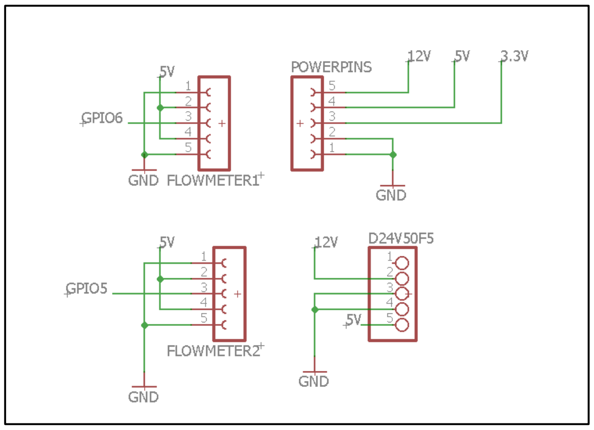


Figure 4.12: Flow Meters, Power Pins, & 5V Step Down Voltage Regulator

4.2.2 Printed Circuit Board Design

Based off of the schematics shown in the previous section, Figure 4.13 below shows the PCB used to power all of the electrical components of the drone. The final board is approximately two inches by three inches. The final design of the board was a 2-layer PCB; the red traces in the figure represent the top of the board while the blue traces represent the back of the board. The traces sending power to the various components and the traces connecting those components to ground are all of a thicker width at 24mil compared to the 16mil of the regular traces sending. Lastly, a ground plane was implemented on the back side of the board in places with empty space, as to ensure a good ground connection on the board.

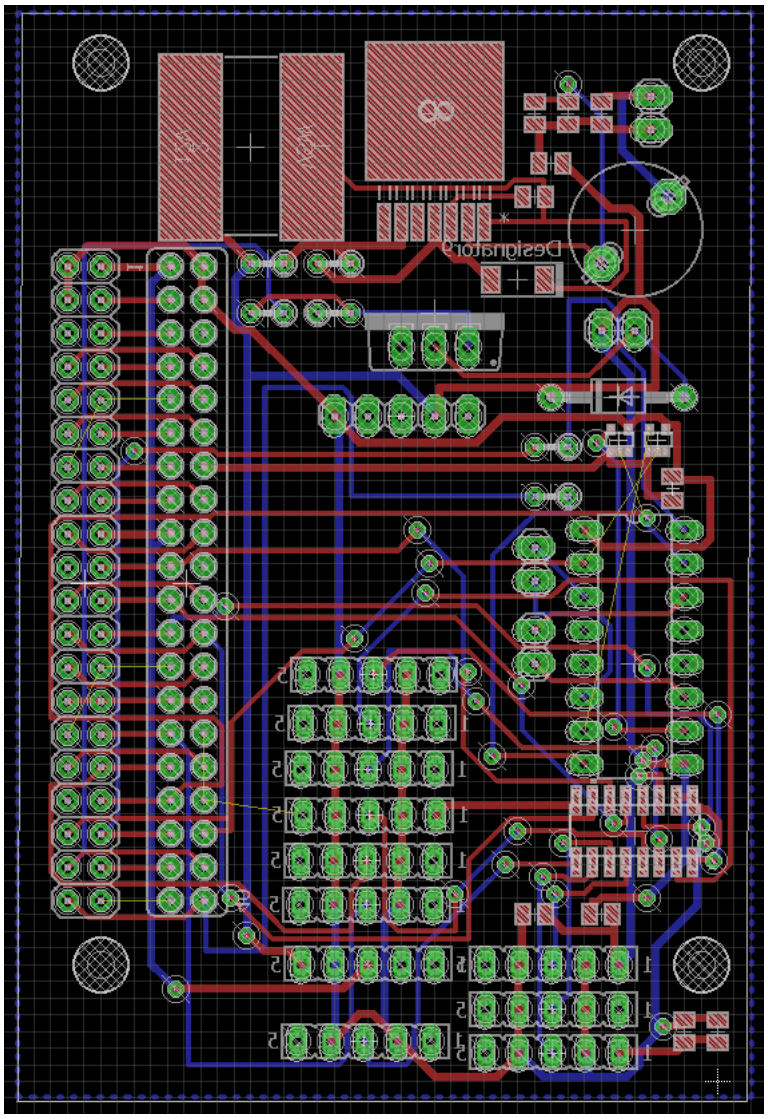


Figure 4.13: Fire Containment Drone PCB Design

4.2.3 System Control

In order to control the system mounted on the drone platform, some sort of microcontroller was needed. The previous team used an Arduino Uno as their microcontroller, however, it has been decided that a Raspberry Pi 3 was to be used for the drone instead. This decision was made to simplify the system and because the team did not believe that any functionality will be lost. The previous team did not use the camera they selected in their end product, so the Raspberry Pi Camera Module V2 was implemented by the team instead. Using the Raspberry Pi in conjunction with the Pi Camera was simpler than using the Arduino with the 3DR camera used by the previous team. Also, the Raspberry Pi and the camera use Python which means that the team would have access to numerous libraries to help with image processing, motor control, and communication. Since there are libraries for all the needed Arduino functions in Python, and due to the simplicity of using Python for the camera and microcontroller, the Raspberry Pi 3 was determined to be a better choice than an Arduino.

Before the final system was created, tests were conducted on the Raspberry Pi. First, the camera was configured to get a video to display on the Raspberry Pi itself. After this, a simple image streaming script was created that would send video to an HTML page on the base computer. Finally, the GPIO, or general purpose input/output, capabilities were tested. This included reading values from the ADC and controlling the motor.

4.3 Software Design

4.3.1 Graphical User Interface

As discussed earlier, the Graphical User Interface needed to be easy to use and have the ability to both control the system and output data, like temperature and video, from the system. First, a mock-up of the interface was created. The team reviewed and adjusted this mockup until it was deemed satisfactory for all of the GUI’s needs. The layout of the GUI was equally important. This interface was designed with ease-of-use in mind.

After the team found an appropriate design, the GUI's visual elements were created and the communication with the drone platform was implemented using HTML and JavaScript respectively.

The final design, pictured in Figure 4.14 below, uses a server built with node.js to host the client, the GUI. This design allows the server to receive 'GET' and 'POST' requests from the client and either send the client data, such as temperatures from the temperature sensors on the drone, or receive data, such as the data that is sent when any of the four command buttons in the GUI are clicked -- turning the pump on and off and lowering and raising the nozzle. With this design, the client can easily access image data from the Raspberry Pi while communicating with the server which provides all of the other data.

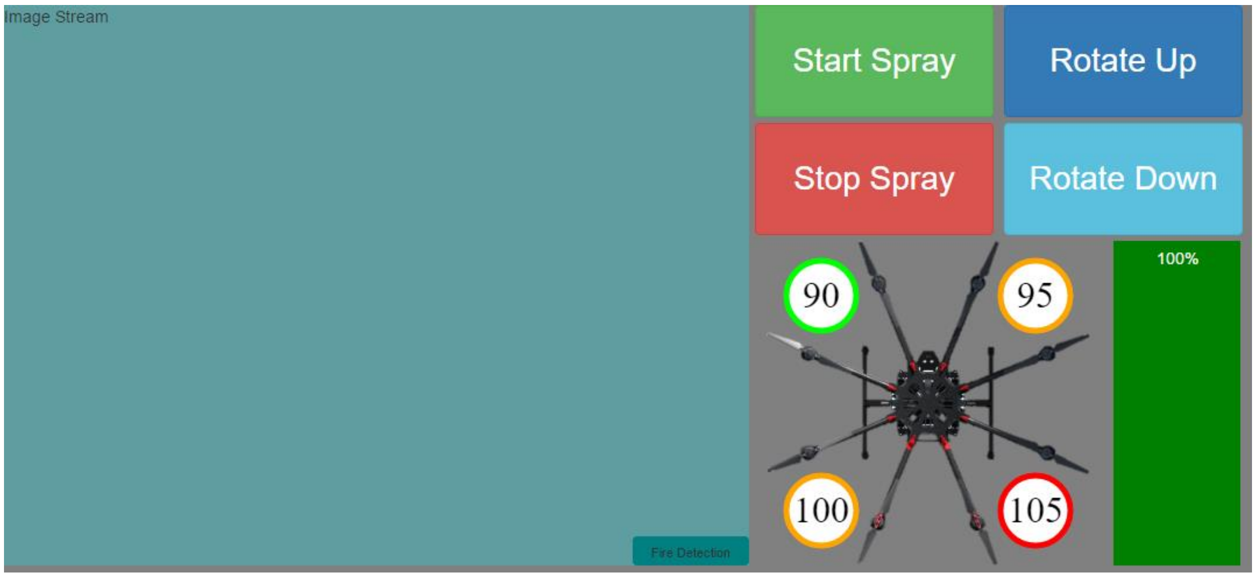


Figure 4.14: Final GUI Design

The team also added in a feature to the GUI which allows the user to toggle a Fire Detection View of the camera. The Fire Detection View uses OpenCV to filter the image so that only colors of fire are visible and then a box is drawn around the largest continuous area of fire. The Fire Detection View's purpose is to aid the user in pinpointing the fire in the camera stream.

In order to bring up the GUI, the user must run the base computer server in the terminal and then load localhost:5002 in an internet browser. Doing so will bring up the GUI and load any immediate data that it needs from the Control Server. The Control Server should be running on the Raspberry Pi before bringing up the GUI on the base computer or it will not connect and work properly.

4.3.2 Communication

There are a large number of messages constantly being sent between the base computer and drone. Since the GUI was implemented in HTML, the team decided to send the messages over a server and client XML HTTP connection. The GUI uses these messages to control the sprayer on the drone while the drone uses the messages to report its current temperature. More specifically, the client receives messages from the server which contain temperature data from the four temperature sensors on the drone every second. The client also sends messages to the server when the user presses any of the five buttons in the GUI which control the pump, nozzle angle, and toggles the Fire Detection View.

Two servers run on the Raspberry Pi, one for the camera's video stream and one for sending and receiving control and status data. The video server runs on port 5001 and will constantly post a jpg image taken from an image queue provided by the image processing algorithm discussed in section 4.3.5. The control server handles requests for data and takes in data from the base computer. The base computer can request both the percent of FireIce remaining, which is just a stored value that is set when the base computer tells the Pi to stop spraying, and the temperature data. When asked for the temperatures, the sensors are read from the ADC chip and converted into Fahrenheit. The control server also processes the following commands from the base computer: Spray, Stop Spray, Rotate Up, Rotate Down, and Toggle Image Processing. Respectively, these control the pump, DoF, and decide whether or not to process the video stream before sending the data. The source code can be found in Appendix B: Source Code.

4.3.3 FireIce Level Sensing

After testing all of the sensors as mentioned in section 2.7.2, the team decided to use timers to estimate the current FireIce level in the tank. This was implemented by having the base computer time how long the pump has sprayed and use that in conjunction with the calculated average flow rate of the FireIce. More specifically, when the user brings up the client, a function is run which fetches the logged current level of the FireIce from the Control Server and displays it in the GUI. Then, when the user selects the “Start” button, a function is run which posts a message to the Control Server to begin spraying, sets the variable *spraying* to be true, and begins the *updateFireIce* function. This function checks if the variable *spraying* is set to true, and if so it calculates the total amount of time this session that the pump has been spraying and uses the flow rate and initial level of the tank to determine the current level of FireIce. If the last height was above 10%, it then updates the height of the tank level bar and label in the GUI. The color of the tank level is also changed to yellow if the height drops below 50% and then red once it is below 25%. If the variable *spraying* is still set to true, a timeout is set to run *updateFireIce* again in 1 second. If the last height of the bar was not above 10%, however, the bar is not updated and an automatic “Stop” command is sent to the Control Server. This was added to prevent the user from accidentally running the pump while there is no FireIce left which could damage it.

When the user selects the “Stop” button, a message is posted to the Control Server to stop the pump, the variable *spraying* is set to false, and the amount of time that the pump has been spraying is logged to be used in further calculations of the height of the bar when the pump is turned on again. The current level of the FireIce is also sent to the Control Server so that it can be saved and accessed later if the client needs to reconnect. All of the code described can be seen in Appendix B: Source Code.

The team decided to have the client keep track of the amount of FireIce in the tank because that would reduce the number of messages sent between the base computer and Raspberry Pi. Keeping the number of messages transmitted low will help to reduce any latency within the Control Server which is one of the team’s goals. However, the team also deemed it necessary to have the server log the level of the FireIce, so this was set to be done only when the pump is turned off as mentioned above.

4.3.4 Temperature Sensing

As previously mentioned, there are four temperature sensors that the client receives data from and displays in the GUI. To do this, when the user brings up the client, after the

FireIce level is initialized, *updateTempData* is run which retrieves the four temperature values from the Control Server. These temperatures are input into a *Draw* function which displays each of the temperatures in a canvas at one of the four corners of the top-view picture of the drone in the GUI. Each of these canvases has a circle around it which is colored based on the temperature in that canvas: less than 95 degrees F is green, 95 to 104 degrees F is orange, and 104 degrees F and above is red. The colors are updated accordingly every time the *Draw* function is run. Upon the function's completion, *updateTempData* sets a timeout to run itself again after 1 second. This updating loop will continue until the client or Control Server is closed. The code for these functions can be seen in Appendix B: Source Code.

4.3.5 Fire Detection Algorithm

The fire detection algorithm is used to pinpoint the exact location of the fire and display it in the GUI. To do this, the algorithm converts the image from BGR values to HSV values. Then, a mask is created by filtering out all of the pixels that are not within a specified range per value. This range was picked to maximize the amount of fire seen while also minimizing stray pixels that make it through the filtering. The algorithm then calculates all of the contours that are in the remaining image. In other words, it finds all of the places where the colors suddenly change. For example, going from red (the fire) to black (the screened out pixels) will create a contour. The algorithm then draws a box around the largest contour to bring the user's attention to the fire. All of these image transformations and manipulations are done using the cv2 library. An example image which was run through the fire detection algorithm can be seen in Figure 4.15 below.

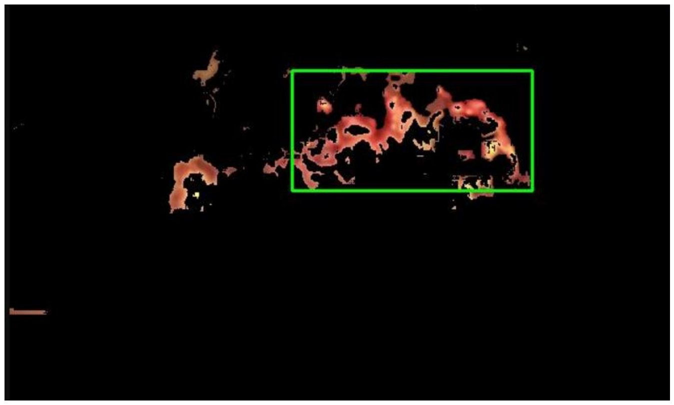


Figure 4.15: Image after running through the fire detection algorithm

5 Results and Discussion

5.1 Mechanical System Results

5.1.1 Spraying System Results

The main requirement for the mechanical system was to be able to propel FireIce to the fire from at least our minimum safe distance to the fire. To ensure that we accomplished this, we set up a test of the entire system. Our team conducted testing at the 2 meter operating condition to assess our calculated values against the experimental as shown in Table 5.1. In addition to this measurement, our team evaluated resulting spray distribution width at the end of the stream, receiving a result of 4 feet, or 1.22 meters.

Height (m)	Angle	Calculated Distance (meters)	Experimental Distance (meters)
2	0	6.18	8.66
2	15	8.82	9.65
2	30	10.90	10.08
2	45	11.25	10.59

Table 5.1: Experimental Distance Results

5.1.2 Tank Results

Our team had calculated that we would get an estimated 3 minutes and 11 seconds of operation out of our tank. From testing we were able to determine that the tank had an experimental operation time of 1 minute and 15 seconds. This value was most likely lower than expected due to imperfections in the tank build, the fact that not all FireIce is usable when air bubbles are introduced, and due to a fluctuation current output during the test. However, knowing this value, we allowed the user to be notified of the FireIce levels when operating the drone via the software implementation on the GUI.

5.2 Electrical System Results

The main electrical requirement for the system, as specified in Chapter 1, was a custom designed PCB to control the electrical power distribution for all the different subsystems. The

PCB and the microprocessor needed to work together to control the on-board mechanisms, sensors, and signals passed between the multiple devices. Another requirement was that a camera system should be present on the system to allow for streaming capabilities and fire detection. The last requirement of the electrical system was that an effective number of communications signals must be implemented to control communications between the drone system and bas

5.2.1 Printed Circuit Board

The custom designed PCB used to power and control the drone as shown in section 4.2.2 was unfortunately not able to be implemented in the final implementation of our system. The reasons as for why the team was not able to implement it were due to electrical failures in the circuitry. The team was not able to fully debug the issues plaguing the PCB, but was able to determine a likely cause. While the PCB was being soldered, mistakes occurred requiring the unsoldering of a couple of key components. It was believed by the team that during the unsoldering process that the slight burning of the board and the resoldering of the small components may have caused the board to short itself, due to the backside being a ground plane. Ideally, the team would have reassessed the design of the PCB to allow for more space for each component to make the soldering of the board easier and proceed to order and solder a new board. However, due to the time constraints placed upon the team, the team decided to recreate the circuitry on the PCB on a protoboard.

The testing done on the protoboard showed a few minor problems with the overall design of the PCB. The first of these problems was the voltage requirement to operate the MOSFET circuit to control the pump. The schematic that was used for the original design was based off of the schematic that the previous team had implemented. It was overlooked by the team that there would be differences in the circuit used between the two different implementations due to the differences between the Arduino used in the previous team's project and the Raspberry Pi used in the current project. In the design of the protoboard's circuitry, a simple schematic was found for the MOSFET and used in place of the schematic shown in section 4.2.1.

The only other problem that was known by the team with the PCB was the connection for pump. It was overlooked that the size requirement for the wires connecting to the pump would be larger than a general purpose wire, used for the other parts of the design. The team corrected this oversight during the implementation of the protoboard. Figure 5.1 below shows a picture of the working protoboard used for the final demonstration of the system.

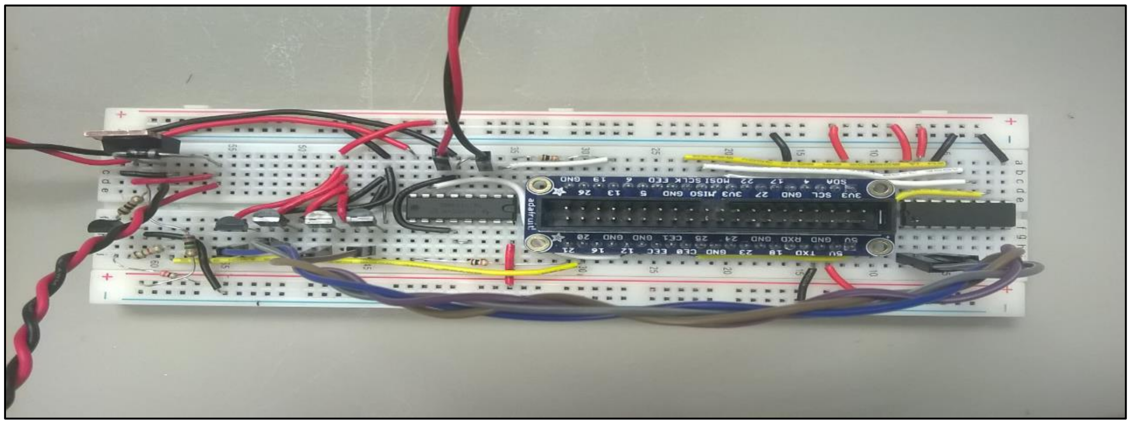


Figure 5.1: Final Protoboard Design and Implementation

5.2.2 FireIce Level Sensing

As part of the sensing system of the drone, the team did not implement FireIce level sensing in any form. Section 2.7.2 details the different type of sensors that were tested and the reasoning behind why the specific type of sensor was not implemented. In lieu of sensing the remaining amount of FireIce in the system, the team decided on a method of estimation using the total time that the system has been dispelling its FireIce solution as an indication of how much of the solution is left. This implementation of FireIce level estimation...

5.2.3 Camera System

The camera system chosen by the team met the requirements as defined in Chapter 1 in that the camera chosen was able to stream video at an effective refresh rate and at a usable resolution for fire detection.

5.2.4 Communications Links

The basic requirement of the communication system on the drone was that the number of communication links between the drone and the base computer was to be minimized while keeping all desired functionality. This requirement was successfully implemented through the use of two links. With the two links, as specified in section 0, the system was able to communicate with the drone all of the necessary commands to operate the fire containment system on the drone, retrieve vital information about the state of the drone such as the readings from the on-board temperature sensors, view the video stream from the drone, and be able to control on the flight of the drone.

5.3 Software System Results

5.3.1 Raspberry Pi and GUI

The Raspberry Pi was first initialized and had its software updated. Next, the camera module was connected and video was streamed to the Pi's screen. This was done to ensure that the camera was working and to get a general idea of the resolution and framerate that could be expected. Once the streaming was working, the GPIO capabilities needed to be tested. The Raspberry Pi was temporarily connected to a breadboard which held an ADC chip that was connected to few other components: LEDs, a servo motor, and several potentiometers. First, the configuration for the communication between the ADC and the Pi through SPI was set up and tested. The potentiometers simulated temperature data that the temperature sensors would output in the final system. The LEDs were then used to familiarize the team with programming the GPIO pins.

Testing was then conducted to make sure that the camera and temperature sensing data would properly be sent to the GUI and displayed. Following this, tests were successfully run to see if the server would receive the commands from the GUI. After this testing was completed, the parameter for the FireIce level sensing estimation algorithm was calibrated and the Raspberry Pi and GUI were ready to be tested within the complete system.

5.3.2 Communication

As previously mentioned, the base communicates with the Raspberry Pi by running a client which is hosted by a server on the Pi. Every time data is requested or sent, it is transmitted via XML HTTP messages. These messages are delivered over the WiFi, so the Raspberry Pi and base computer must always be connected to the same WiFi. The strength of the WiFi connection could result in delays in the messages and camera streaming to the base computer, but this will be tested as soon as the team has the entire system built. Assuming that there is a good WiFi connection, the XML HTTP messages will be reliable as HTTP messages already deal with dropped packages and have error detection built into it.

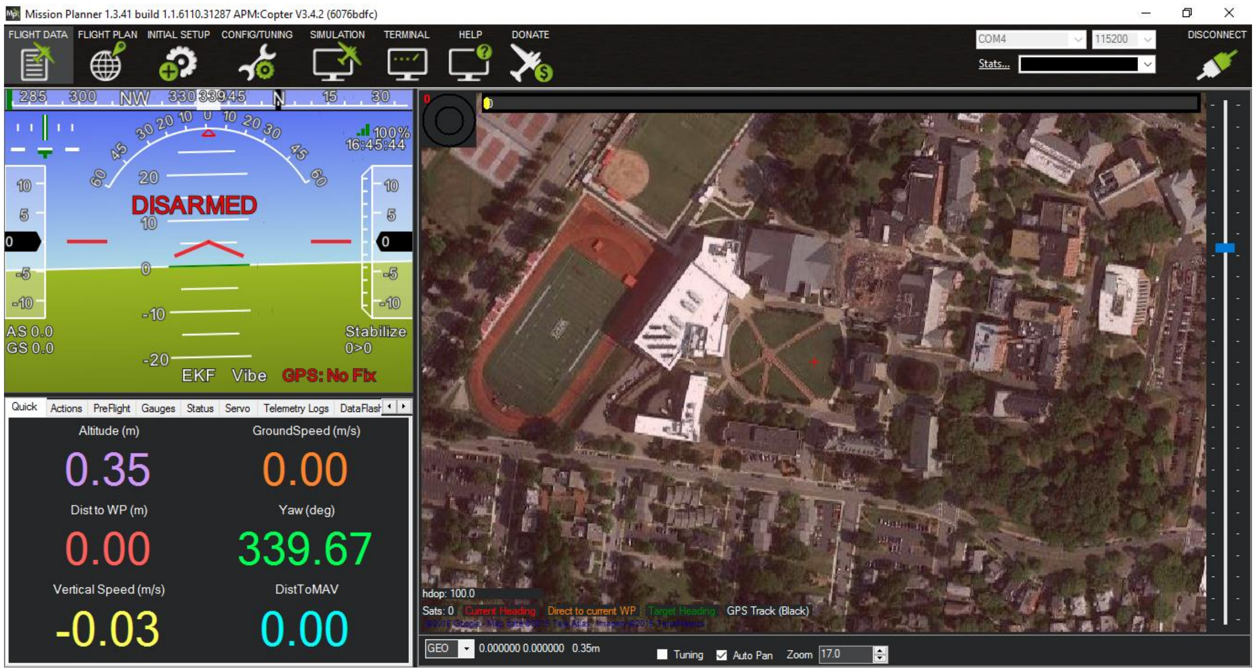
When it comes to further projects, it would be best to integrate a communication method that would allow the drone and base computer to be off of any WiFi. For this project, however, a WiFi connection was needed for other purposes, so it was best to keep the communication lines to a minimum and utilize what the team already had.

5.3.3 Fire Detection Algorithm

The Fire Detection Algorithm was calibrated and tested with multiple fire sources. The first tests were on videos of boat fires. These tests were relatively successful, screening out most of the undesired colors from the image stream. The reflection of the screen did cause some uncertainty in the testing results which is why the team then moved on to testing with candles. Upon pointing the camera at the fire, the algorithm accurately captured all of the colors of the fire while screening out the rest. After a few more tests in different lighting environments, the team deemed the candle testing to be successful, and the algorithm was ready to be fully integrated into the entire system.

5.3.4 Mission Planner

In order to calibrate the Pixhawk's compass and magnetometer, the team used Mission Planner, a ground station application provided by ArduPilot. Every time the drone is used, it should be properly calibrated in order to optimize flight. Although Mission Planner allows for users to send the drone target locations, the team will not have much use for that, however, Mission Planner does allow for the calibration of the Pixhawk and viewing of the preflight warnings and errors which are essential to the flight safety of the drone. Figure 5.2 below shows Mission Planner's flight data interface which displays the warnings and errors as well as the drone's position and flight metrics.



AS 0.0
GS 0.0
Stabilize 0>0

Altitude (m) 0.35
GroundSpeed (m/s) 0.00
Dist to WP (m) 0.00
Yaw (deg) 339.67
Vertical Speed (m/s) -0.03
DistToMAV 0.00

hdop: 100.0
Sats: 0

0.000000 0.000000 0.35m
Zoom 17.0

Figure 5.2: Mission Planner Flight Data Interface

6 Conclusions and Recommendations

The goal of the Fire Containment Drone was to create a UAV system in order to aid in the containment and control of offshore vessel fires. This was completed by determining specific requirements that must be addressed in order to have a functional and effective system. In conclusion, the team was able to meet all the design requirements after researching the different choices and choosing the best options. Although the team ran into many different changes over the course of the project, the final testing of the system delivered desirable results that allows for future opportunities and further advancement,

Mechanically, the system performed as expected, if not better. The system was designed with a safety factor of 1.5 for the minimum safe distance of the drone away from the fire. This led us to need an output that could span a minimum of 6.18 meters using a solution of 20% FireIce and 80% water. From our testing, we found that our system was capable of 8.66 meters at the most critical case (elevation of 2 meters, pitch of 0 degrees), exceeding our expectations. In addition, we were able to achieve a 1.22 meter, or 4 feet, spray coverage, which is satisfactory for a path of egress.

Future recommendations for the mechanical side include increasing the FireIce holding capacity and increasing the pressure output by implementing a more powerful pump. However, the current system is as optimized as possible given our weight restrictions, so a drone that is capable of carrying a higher payload would be required. By implementing a better pump, it would be able to ideally use a fluidic oscillator which would achieve similar results to the solid stream nozzle, but also be able to fluctuate the orientation of the stream mechanically.

Electrically, although the PCB did not go as planned, the team was still able to effectively create the same result on a protoboard that could be put on the system. The different circuits on the protoboard were able to produce the results allowing the software and mechanical portions of the system to communicate. The Raspberry Pi was a good microprocessor choice that was very universal and fulfilled many of the requirements the team needed. With built-in WiFi, GPIO pins, and high processing power, the team was able to use this single device for electrical and software needs.

Future recommendations for the electrical side include having a functional working PCB that had all the required circuits and components as well as some extra space for improvements and changes. Due to time restraints the team was not able to order a second version of the PCB with the fixed problems. The original PCB was also very small and made it very difficult to solder and change out components as needed. Increasing space on the physical face of the board should be a priority. In addition, a better way to detect the FireIce levels in the storage tank should be implemented. The team choose to go with a software implementation based on time, however, a sensor measurement would be more reliable.

Future recommendations for the software would be to move more towards a completely autonomous system. The team started working on portions such as the fire detection algorithm that could be use in an autonomous system. In addition, the team used a WiFi router to host the communication between the base computer and the system on the drone platform. Although this worked well, it has very limited range and would not be ideal for land/air communication. The implementation of higher range devices over radio signal is more preferred.

Finally, in general, the team was not able to test the system on the drone in a flight test due to time restraints and heavy delays. These test should be carried out moving forward to get a clear understanding of what can be improved on and precisely how effective the system performs under flight conditions. A live test fire should also be performed to evaluate the performance of the FireIce fire retardant.

7 References

OpenCV Documentation. (2016). Retrieved from http://opencv.org

Teledyne DALSA. *"CCD vs. CMOS"* Teledyne DALSA Inc., n.d. Web. 15 Sept. 2017. <http://www.teledynedalsa.com/imaging/knowledge-center/appnotes/ccd-vs-cmos/>

Floreano, D., & Wood, R. J. (2015). "Science Technology and the Future of Small Autonomous Drones" Nature. 21 Sept. 2016. Web. <http://www.nature.com/nature/journal/v521/n7553/full/nature14542.html>

Hansen, C. (2015). "How To Measure The Level Of Viscous Liquids." *APG*. Web. 15 Sept. 2017. <https://www.apgsensors.com/about-us/blog/how-to-measure-the-level-of-viscous-liquids>.

SightLogix., n.d. "How Thermal Cameras Work" Web. 21 Sept. 2017. <http://www.sightlogix.com/how-thermal-cameras-work/>.

Kim, G. (201A study of fluidic oscillators as an alternative pulsed vortex generating jet actuator for flow separation control 1).

Lockman, Arthur, and Tucker Haydon. *De-Mining with UAVs*. Rep. Worcester: Worcester Polytechnic Institute, 2016. Print.

MAVLink micro air vehicle communication protocol. Retrieved from http://qgroundcontrol.org/mavlink/start

MAVLink python bindings. (2016). Retrieved from http://qgroundcontrol.org/mavlink/pymavlink

National Fire Protection Association., Society of Fire Protection Engineers., and Books24x7 Engineering Pro Collection. (2002) *SFPE Handbook of Fire Protection Engineering*. National Fire Protection Agency. Web.

National Research Council, Staff, Complete Ebrary Academic, and Halon National Research Council . Naval Studies Board. Committee on Assessment of Fire Suppression Substitutes and Alternatives to. *Fire Suppression Substitutes and Alternatives to Halon for U.S. Navy Applications*. Washington, D.C: National Academy Press, 1997. Print.

NFPA. (2016). NFPA 1405: Guide for land based departments that respond to marine vessel fires (6th ed.) National Fire Protection Agency.

NFPA. (2015) "All About Fire." National Fire Protection Association. Web. 21 Sept. 2016. <http://www.nfpa.org/news-and-research/news-and-media/press-room/reporters-guide-to-fire-and-nfpa/all-about-fire>

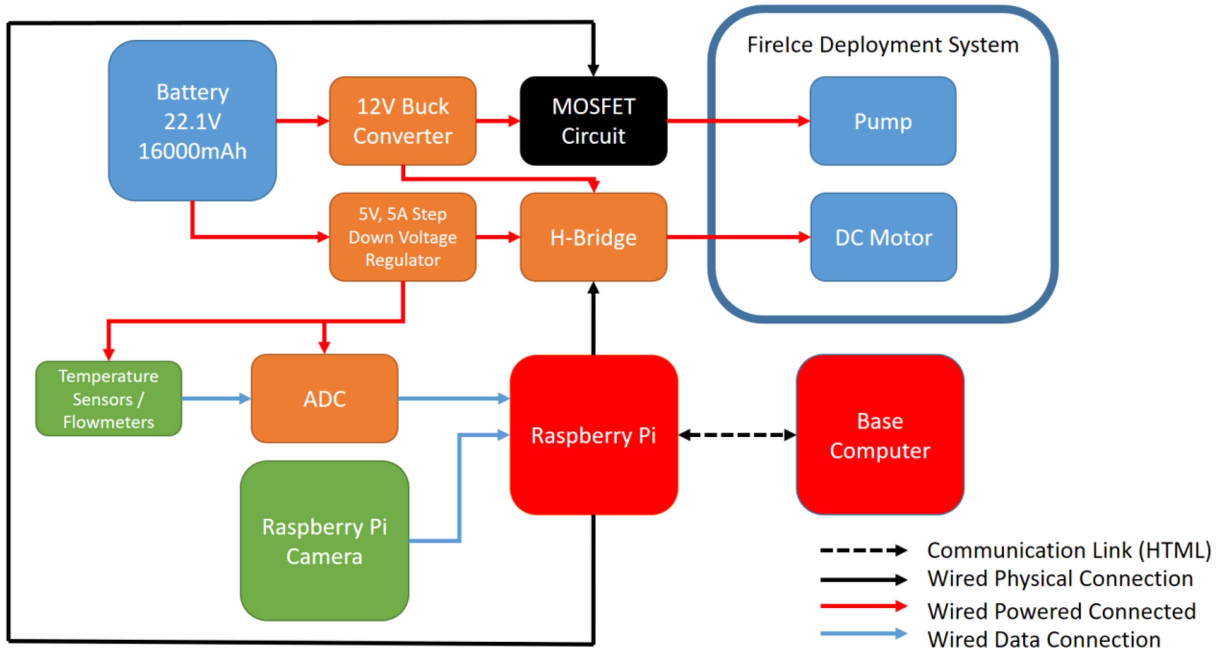
Pahlaven, K., & Krishnamurthy, P. (2013). Principles of Rireless Access and localization Wiley.

Peterson, C., Neilan, T., Rubenstein, D., & Toribio, B. (2016). Fire Containment Drone. Worcester Polytechnic Institute. Report. 28 Aug. 2016.

Weyandt, N., & Janssens, M. (2008). Fire extinguisher performance evaluation with GelTech solutions inc.'s FireIce water additive on class 2-A and 40-A cribs and A ten-tire fire in general accordance with UL 711. (). San Antonio, TX: SouthWest Research Institute.

XBee® DigiMesh® 2.4. Retrieved from https://www.digi.com/products/xbee-rf-solutions/modules/xbee-digimesh-2-4#specifications

8 Appendix A: System Diagram



9 Appendix B: Source Code

Raspberry Pi (Python)

```
import cv2
from PIL import Image
from http.server import BaseHTTPRequestHandler, HTTPServer
import io
import time
import cgi
from multiprocessing import Queue, Process, Value
import signal
from picamera import PiCamera
from picamera.array import PiRGBArray
from time import sleep
import numpy as np
import RPi.GPIO as GPIO
import os
import spidev

"""
Written by: John Lomi
"""

#define pins
motor1Inv = 12
motor1 = 13
motor1En = 19
pumpPin = 20

#setup GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(motor1Inv, GPIO.OUT)
GPIO.setup(motor1, GPIO.OUT)
GPIO.setup(motor1En, GPIO.OUT)
GPIO.setup(pumpPin, GPIO.OUT)

GPIO.output(pumpPin, 1)
GPIO.output(motor1En, 0)

#setup SPI
spi = spidev.SpiDev()
spi.open(0,0)

#setup camera
capture = None
startQ = Queue(maxsize=3)
endQ = Queue(maxsize=3)


class CamHandler(BaseHTTPRequestHandler):
    """Handle video streaming

    Arguments:
    BaseHTTPRequestHandler -- The handler for the HTTP requests

    This class handles the streaming of the video to all clients that submit
    a 'GET' request
    """

    def do_GET(self):
        """Post a jpg repeatidly to all connected clients to create a
video"""
        if self.path.endswith('.mjpg'):
            self.send_response(200)
            self.send_header('Content-type','multipart/x-mixed-replace;
boundary=--jpgboundary')
            self.end_headers()

            while True:
                try:
                    frame = endQ.get(block=True, timeout=None)
                    jpg = Image.fromarray(frame, 'RGB')
                    tmpFile = io.BytesIO()
                    jpg.save(tmpFile, 'JPEG')
                    self.wfile.write(bytes("--jpgboundary", 'UTF-8'))
                    self.send_header('Content-type', 'image/jpeg')
                    self.send_header('Content-length',
str(bytes(tmpFile.getbuffer().nbytes), 'UTF-8'))
                    self.end_headers()

                    jpg.save(self.wfile, 'JPEG')

                    print("Image sent")

                except KeyboardInterrupt:
                    print("KeyboardInterrupt caught")
                    break
            return

        if self.path.endswith('.html'):
            print("path ends with .html")
            self.send_response(200)
            self.send_header('Content-type', 'text/html')
            self.end_headers()

            self.wfile.write(bytes('<html><head></head><body>', 'UTF-8'))
            self.wfile.write(bytes('<img
src="http://192.168.1.130:5001/cam.mjpg"/>', 'UTF-8'))
            self.wfile.write(bytes('</body></html>', 'UTF-8'))

            return


class CtlHandler(BaseHTTPRequestHandler):
    """Handle control and status communications

    Arguments:
```

```
    BaseHTTPRequestHandler -- The handler for the HTTP requests

    This class handles requests for statuses and commands for system control
    """

    def do_GET(self):
        """Send information to base computer including FireIce level and
temperature data"""
        global tankPct
        print("CtlHandler: do_GET")
        self.send_response(200)
        self.send_header('Content-type', 'text/plain')
        self.send_header('Access-Control-Allow-Origin', '*')
        self.end_headers()

        if self.path.endswith('getPct'):
            #send back tank percentage
            self.wfile.write(bytes(str(tankPct.value), 'UTF-8'))
        elif self.path.endswith('getTemp'):
            #measure temp sensors
            temp_data = getTemp()
            #send back the data
            self.wfile.write(bytes(temp_data, 'UTF-8'))

        return

    def do_POST(self):
        """Process commands to the Pi including Spray, Rotation, and Image
Processing"""
        global pImg, tankPct
        print("CtlHandler: do_POST")
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length)

        self.send_response(200)
        self.send_header('Access-Control-Allow-Origin', '*')
        self.end_headers()

        if self.path.endswith('setPct'):
            #update tank percentage
            post_data = float(post_data)
            tankPct.value = post_data
            print("---------- {}".format(tankPct.value))
        else:
            post_data = int(post_data)

            #switch through data types
            if post_data == 1:
                #Start spraying
                print("---------- Spray")
                startSpray()
                pass
            elif post_data == 2:
                #Stop spraying
                print("---------- Stop Spray")
                stopSpray()
                pass
            elif post_data == 3:
                #Rotate Up
                print("---------- Rotate Up")
                rotateUp()
                pass
            elif post_data == 4:
                #Rotate Down
                print("---------- Rotate Down")
                rotateDown()
                pass
            elif post_data == 5:
                #toggle image processing
                pImg.value = 1 ^ pImg.value
                print("---------- Toggle Img Processing")
            else:
                print("Unrecognized post_data")
        return


def getTemp():
    """Return temperature data from the sensors using the ADC and SPI"""
    temp1 = toTemp(readADC(0))
    temp2 = toTemp(readADC(1))
    temp3 = toTemp(readADC(2))
    temp4 = toTemp(readADC(3))

    return '{}\n{}\n{}\n{}'.format(temp1, temp2, temp3, temp4)


def readADC(chan):
    """Read data from the ADC on the specified channel

    Arguments:
    chan -- the ADC channel to use
    """

    adc = spi.xfer2([1, (8+chan)<<4, 0])
    data = ((adc[1]&3) << 8) + adc[2]
    return data


def toTemp(data):
    """Convert ADC data to a Farenheight temperature

    Arguments:
    data -- the value to be converted
    """

    return int((data * 297)/float(1023))


def startSpray():
    """Turn on the pump to start spraying"""
    GPIO.output(pumpPin, 0)
    pass
```

```
def stopSpray():
    """Turn off the pump to stop spraying"""
    GPIO.output(pumpPin, 1)
    pass


def rotateUp():
    """Turn the DoF motor CCw"""
    GPIO.output(motor1, 1)
    GPIO.output(motor1Inv, 0)

    GPIO.output(motor1En, 1)
    sleep(.1)
    GPIO.output(motor1En, 0)
    pass


def rotateDown():
    """Turn the DoF motor CW"""
    GPIO.output(motor1, 0)
    GPIO.output(motor1Inv, 1)

    GPIO.output(motor1En, 1)
    sleep(.1)
    GPIO.output(motor1En, 0)
    pass


def putImage():
    """Grab image from camera and put into a queue"""
    global shutdown, frameQ
    camera = PiCamera()

    camera.framerate = 15
    camera.resolution = (640, 480)

    rawCap = PiRGBArray(camera, size=(640, 480))

    #allow camera to warm up
    sleep(.5)

    try:
        for frame in camera.capture_continuous(rawCap, format="bgr",
                                               use_video_port=True):
            frame = frame.array

            #try to add image to queue, skip if timeout
            try:
                startQ.put(frame, block=False)
            except:
                pass

            #clear stream
            rawCap.truncate(0)

    except KeyboardInterrupt:
        print("KeyboardInterrupt Caught")
        #cleanup
        camera.close()
        cv2.destroyAllWindows()


def processImage():
    """Process the frames from the camera and put them in a queue to be
transmitted"""
    try:
        while True:
            try:
                image = startQ.get(block=True, timeout=1)
            except:
                pass
            else:
                if(pImg.value):
                    #convert to HSV
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

                    #define mask boundaries
                    # [0, 178, 178]  [32, 255, 255]
                    lower = np.array([0, 125, 125], dtype='uint8')
                    upper = np.array([32, 255, 255], dtype='uint8')

                    #mask image
                    mask = cv2.inRange(image, lower, upper)
                    #image = cv2.bitwise_and(image, image, mask = mask)

                    #find contours
                    contours = cv2.findContours(mask, cv2.RETR_EXTERNAL,
                                                cv2.CHAIN_APPROX_SIMPLE)[-2]

                    #find the largest contour
                    if(len(contours)):
                        maxLen = 0
                        maxInd = 0
                        for i in range(0, len(contours)):
                            contLen = len(contours[i])
                            if(contLen > maxLen):
                                maxLen = contLen
                                maxInd = i

                        #create box around largest contour
                        x,y,w,h = cv2.boundingRect(contours[maxInd])

                        image = cv2.cvtColor(image, cv2.COLOR_HSV2RGB)

                        cv2.rectangle(image, (x,y), (x+w, y+h), (0,255,0), 2)
                    else:
                        image = cv2.cvtColor(image, cv2.COLOR_HSV2RGB)

                else:
                    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)


                #put image in final queue
                endQ.put(image, block=True, timeout=None)
    except KeyboardInterrupt:
        pass


def streamServer():
    """Start video streaming server"""
    try:
        server = HTTPServer(('', 5001), CamHandler)
        print("5001 started")
        server.serve_forever()
    except KeyboardInterrupt:
        server.socket.close()


def controlServer():
    """Start control server"""
    try:
        server = HTTPServer(('', 5002), CtlHandler)
        print("5002 started")
        server.serve_forever()
    except KeyboardInterrupt:
        server.socket.close()


def main():
    global tankPct
    global pImg
    tankPct = Value('f', 100.0)
    pImg = Value('i', 0)

    key = cv2.waitKey(1) & 0xFF

    #initialize image processing processes
    putP = Process(target=putImage)
    imgP1 = Process(target=processImage)

    #initialize server processes
    streamP = Process(target=streamServer)
    controlP = Process(target=controlServer)

    #start all processes
    putP.start()
    imgP1.start()
    streamP.start()
    controlP.start()

    #loop until ctrl-c
    try:
        while True:
            pass
    except KeyboardInterrupt:
        pass



    #end processes
    print("Ending Processes...")
    GPIO.cleanup()
    imgP1.join()
    putP.join()
    streamP.join()
    controlP.join()


if __name__ == '__main__':
    main()
```

Base Computer Server (JavaScript)

```
/*
 * Written by Kyle Young
 * Last edited: 3/15/2017
 *
 * This code will run the GUI for the fire containment drone on localhost
port 5002.
 * It also provides the appropriate libraries and images so that the GUI is
properly displayed.
 */

var http = require('http')
  , fs   = require('fs')
  , url  = require('url')
  , qs   = require('querystring')
  , path = require('path')
  , port = 5002;

// The function run every time a page is loaded on the server
var server = http.createServer (function (req, res) {
  var uri = url.parse(req.url, true)

  //function sendIndex(res, movieList,timer, query, numRes)
  switch( uri.pathname ) {
    case '/':
      sendFile(res, 'final_gui.html', 'text/html')
      break
    case '/final_gui.html':
        sendFile(res, 'final_gui.html', 'text/html')
        break
    case '/img/drone.png':
      sendFile(res, 'public/img/drone.png', 'image/png')
      break
    case '/js/scripts.js':
      sendFile(res, 'public/js/scripts.js', 'text/javascript')
      break
    case '/style.css':
        sendFile(res, 'public/css/style.css', 'text/css')
        break
    case '/bootstrap.min.css':
      sendFile(res, 'public/css/bootstrap.min.css', 'text/css')
      break
    case '/bootstrap.min.css.map':
      sendFile(res, 'public/css/bootstrap.min.css.map', 'text/css')
      break
    default:
      res.end('404 not found')
  }
});

// Listens on environment's port or port 5002
server.listen(process.env.PORT || 5002)
console.log('listening on 5002')
var temps = [75,80,85,70]


// sends files to client when requested
function sendFile(res, filename, contentType) {
  contentType = contentType || 'text/html'

  fs.readFile(filename, function(error, content) {
    res.writeHead(200, {'Content-type': contentType})
    res.end(content, 'utf-8')
  })

}
```

Graphical User Interface (HTML)

```
<!--
Written by Kyle Young
Last Edited: 3/15/2017

This code will generate the GUI for the fire containment drone. If it does not connect to the
Control Server at the URL in the img tag,
it will not load the display correctly.
-->

<html>
<head>
	<title>Fire Containment Drone</title>
	<meta charset="utf-8">
	<meta http-equiv="X-UA-Compatible" content="IE=edge">
	<meta name="viewport" content="width=device-width, initial-scale=1">
	<link rel="stylesheet" type="text/css" href="style.css">
	<link rel="stylesheet" type="text/css" href="bootstrap.min.css">
</head>

<body style="background-color:grey;">
	<script src="js/scripts.js"></script>
	<div class="content">
```

```
			<img src="http://192.168.1.131:5001/cam.mjpg" alt="Stream Picture"
width=640 height=480 style="float:left">
			<!-- <div id="picture">Image Stream</div> -->
			<input type="button" name="downBtn" id="fireDetect" value="Fire Detection"
class="enable btn" onClick="toggle();">
		</div>

		<div class="sidebar">
			<div>
				<input type="button" name="sprayBtn" id="spray" value="Start Spray"
class="enable btn btn-success" onClick="sendData(1);">

				<input type="button" name="upBtn" id="up" value="Rotate Up"
class="enable btn btn-primary" onClick="sendData(3);">
			</div>
			<div>
				<input type="button" name="stopBtn" id="stop" value="Stop Spray"
class="enable btn btn-danger" onClick="sendData(2);">

				<input type="button" name="downBtn" id="down" value="Rotate Down"
class="enable btn btn-info" onClick="sendData(4);">
			</div>
			<div>
				<img id="droneImg" id="droneImage" src="img/drone.png" alt="Image of
Drone" width='278px' height='278px' style="float:left">
				<canvas id="temp0" width='278px' height='278px'></canvas>
				<canvas id="temp1" width='278px' height='278px'></canvas>
				<canvas id="temp2" width='278px' height='278px'></canvas>
				<canvas id="temp3" width='278px' height='278px'></canvas>

				<div id="myProgress">
				  <div id="myBar">
				    <div id="label">100%</div>
				  </div>
				</div>
			</div>
		</div>
		<script>
			initializeFireIceLevel()

			window.beforeunload = closingCode;

			// Draw([90,95,100,105]); // can be run instead of updating temp data in
initializeFireIceLevel to display example data
```

```
        </script>
</body>
</html>
```

Graphical User Interface (JavaScript)

```
/*
 * Written by Kyle Young
 * Last Edited: 3/15/2017
 *
 * This file contains all of the scripts used in the fire containment drone
GUI:
 *     - Sending data
 *     - Updating temp data
 *     - Displaying temps
 *     - Initializing the FireIce level
 *     - Updating the FireIce level
 */

//================= Variables
=============================================================================
==================================
var spraying = 0; //variable to keep track of whether the drone is currently
spraying or not. Used for FireIce update
var height = 100; //starting height of the fireIce bar
var initialHeight = 100; //initial height of the FireIce bar retrieved from
the control server
var tankSeconds = 75; //seconds until tank runs out of FireIce
var percentPerSecond = 100/tankSeconds; //percentage that the bar should drop
per second
var startTime = 0; //initializes the starting spray time to 0
var loggedTime = 0; //the amount of time in the past during this session that
the pump has sprayed
var lastHeight = 100; //previous height of the FireIce bar
var fireDetect = 0; //toggle for fire detection algorithm button
var requestTemp = 1; //variable used to stop the client from requesting
temperatures from the server after the client is closed

//================= Functions
=============================================================================
==================================
var IP_ADDRESS = "192.168.1.131";

/*
 * Send data to the control server
 * @param num - the number to be sent based on the button pressed
 */
function sendData ( num ) {
  var oReq = new XMLHttpRequest();

  oReq.open("POST", "http://" + IP_ADDRESS + ":5002", true);

  oReq.send(num);

  console.log("Sent: " + num);
  if(num == 1){ //if start spraying
    spraying = 1;
    console.log("spraying="+spraying)

    //set start time for spraying
    startTime = new Date().getTime()/1000;

    //start updating FireIce level
    updateFireIce();
  }
  else if(num == 2){ //if stop spraying
    spraying = 0;
    console.log("spraying="+spraying)

    //calculate time since started spraying
    var addedTime = new Date().getTime()/1000 - startTime;

    //update total past time sprayed since session started
    loggedTime = loggedTime + addedTime;

    console.log("loggedTime = " + loggedTime);

    //update control server's logged FireIce level
    var oReq2 = new XMLHttpRequest();
    oReq2.open("POST", "http://" + IP_ADDRESS + ":5002/setPct", true);
    oReq2.send(height.toString());
  }
}

/*
 * Toggle the color for the fire detection algorithm button so that the user
knows if it is currently selected or not
 */
function toggle(){
    //send fire detection message to control server
    sendData(5);

    var fireButton = document.getElementById("fireDetect");

    if(fireDetect == 0){ //if fire detecting
        fireButton.style.backgroundColor = "#030D73";
        fireButton.style.color = "white";
        fireDetect = 1;
    }
    else { //if not fire detecting
        fireButton.style.backgroundColor = "teal";
        fireButton.style.color = "inherit";
        fireDetect = 0;
    }
}

/*
 * Code to run when the page closes
 */
function closingCode(){
    requestTemp = 0;
    return null;
}

/*
 * Updates the temperature data by requesting it from the control server
 */
function updateTempData (){
    console.log('Updating temps');
    if(requestTemp == 1){
        function reqListener(){
            var temp = this.responseText.split('\n');

            //draw temps
            Draw(temp)

            //set timeout for another temp update
            setTimeout(updateTempData,1000)
        }

        var oReq = new XMLHttpRequest();

        oReq.addEventListener("load", reqListener)

      oReq.open("GET", "http://" + IP_ADDRESS + ":5002/getTemp", true);
      oReq.send(1);
    }
}

/*
 * Draws the temperatures onto the top-view of the drone
 * @param temps - an array of temperature strings
 */
function Draw(temps){
    var img = document.getElementById("droneImg");
    for(var i = 0; i < 4; i++){
      var cnvs = document.getElementById('temp'+ i);

      cnvs.style.position = "absolute";
      cnvs.style.left = img.offsetLeft + "px";
      cnvs.style.top = img.offsetTop + "px";

      var ctx = cnvs.getContext("2d");
      ctx.beginPath();

      ctx.font = "32px serif";

      //put text in ring
      if(i == 0){ //front left
        ctx.arc(47, 47, 30, 0, 2 * Math.PI, false);
        ctx.fillStyle = 'white';
          ctx.fill();
          ctx.fillStyle = 'black';
          ctx.textAlign = 'center';
        ctx.fillText(temps[i], 47, 57);
      }
      else if(i == 1){ //front right
        ctx.arc(231, 47, 30, 0, 2 * Math.PI, false);
        ctx.fillStyle = 'white';
          ctx.fill();
          ctx.fillStyle = 'black';
          ctx.textAlign = 'center';
        ctx.fillText(temps[i], 231, 57);
      }
      else if(i == 2){ //back left
        ctx.arc(47, 231, 30, 0, 2 * Math.PI, false);
        ctx.fillStyle = 'white';
          ctx.fill();
          ctx.fillStyle = 'black';
          ctx.textAlign = 'center';
        ctx.fillText(temps[i], 47, 241);
      }
      else{ //back right
        ctx.arc(231, 231, 30, 0, 2 * Math.PI, false);
        ctx.fillStyle = 'white';
          ctx.fill();
          ctx.fillStyle = 'black';
          ctx.textAlign = 'center';
        ctx.fillText(temps[i], 231, 241);
      }


      ctx.lineWidth = 5;

      //adjust colors of ring based on temp
      if(temps[i] < 95){
        ctx.strokeStyle = '#00ff00';
      }
      else if(temps[i] < 104){
        ctx.strokeStyle = 'orange';
      }
      else{
        ctx.strokeStyle = 'red';
      }

      ctx.stroke();
    }
}

/*
 * Updates the FireIce bar in the GUI based on the change of time since
spraying began and the initial level
 */
function updateFireIce(){
    if(spraying == 1){ //if currently spraying
        var timeSprayed = new Date().getTime()/1000 - startTime + loggedTime;
        height = Math.max(Math.round((initialHeight - (percentPerSecond *
timeSprayed))*100)/100,0);
        if(lastHeight > 10){ //if last height was great enough, continue to
spray
            var level = document.getElementById("myBar");

            var barHeight = 278; //should be dynamically allocated, but was
unable to
```

```
            // calculate height of bar
            level.style.height = height + '%';
            document.getElementById("label").innerHTML = height * 1 + '%';
            level.style.marginTop = barHeight -
Math.floor(barHeight*height/100)+'px';

            console.log(height)
            lastHeight = height;

            // choose appropriate color for bar
            if(height > 50){
                level.style.backgroundColor = "green";
            }
            else if(height > 25){
                level.style.backgroundColor = "orange";
            }
            else{
                level.style.backgroundColor = "red";
            }

            if(spraying){
                console.log("adding updateFireIce to stack")
                setTimeout(updateFireIce,1000);
            }
        }
        else{ //stop spraying
            sendData(2);
        }
    }
}

/*
 * Retrieve the initial FireIce level from the control server
 */
function initializeFireIceLevel(){
    function reqListener(){
        var temp = this.responseText.split('\n');
        initialHeight = parseInt(temp);

        var level = document.getElementById("myBar");

        var barHeight = 278; //should be dynamically allocated, but wasn't
working

        // calculate height for bar
        level.style.height = initialHeight + '%';
        document.getElementById("label").innerHTML = initialHeight * 1 + '%';
        level.style.marginTop = barHeight -
Math.floor(barHeight*initialHeight/100)+'px';

        console.log(initialHeight)
        lastHeight = initialHeight;

        // choose appropriate color for bar
        if(initialHeight > 50){
            level.style.backgroundColor = "green";
        }
        else if(initialHeight > 25){
            level.style.backgroundColor = "orange";
        }
        else{
            level.style.backgroundColor = "red";
        }

        updateTempData();
    }

    var oReq = new XMLHttpRequest();

    oReq.addEventListener("load", reqListener)

  oReq.open("GET", "http://" + IP_ADDRESS + ":5002/getPct", true);

  oReq.send();
  console.log("initializing FireIce Level");
}
```