The Maze Zone Drone

Ehsan Falaki, Tanner Foster,
Matt Szewczyk, Justin Yuen

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

***Abstract* — This paper presents the design methodology used to realize a multi-system electrical and computer engineering project, with a unique implementation to 3D maze solving. The Maze Zone Drone is an autonomous quadcopter, whose objective is to fly and locate points of interest on a maze. It captures a stable image, which is then transmitted to a ground station computer located on the ground vehicle. This system's objective is to analyze the image and solve the maze. To top it off, the ground vehicle will then take this maze solution to successfully navigate the maze on the first attempt. The focus of this project is to achieve seamless integration of all components and systems. Potential applications include functioning as a parking spot locator and even military obstacle avoidance.**

***Index Terms* — Aircraft Navigation, Communication Systems, Image Analysis, Maze, Object Detection, Vehicle Routing.**

I. Introduction

Conceived with a strong interest to utilize a drone, the Maze Zone Drone project takes traditional maze solving, and tests the intellectual reaches of the design team members. The main objective is to help a ground vehicle solve a physical maze on the first attempt, with the assistance of a drone and image processing above. To achieve a perfectly autonomous system (with few hardcoded values), seamless integration of each sub system was required. With each added component and line of code, the chance for failure increased. Proactive research and rapid prototype testing were practiced to minimize risk.

II. System Overview

The aerial vehicle—the Maze Zone Drone itself—is a quadcopter designed from scratch. A few key specifications include that it should fly for a minimum of ten minutes, with wireless communication up to thirty meters, along with altitude accuracy. It features DC motors, flight controllers, flight sensors, electronic speed controllers, a transmitter/receiver, camera, and power supply system. In addition to manual control, autonomous stabilization and hovering capabilities in unpredictable environments needed to be perfected. A ground control station with mission planner and mavproxy assisted in initial testing and flight control scripts, a key design point.

Image processing is the most software intensive component of this project, utilizing Python, C, C#, and OpenCV languages. After a stable image is captured by the aerial vehicle, several images are output in series to ensure the ground vehicle can solve the maze. It entails color filtering, object detection, edge detection, maze solving, and solution translation. The maze solving algorithm itself is quite simple, detecting two wall components by a morphological operation, with the 'AND' region of these being the solution path. Its only constraint is that the maze must by one-to-one, having only one solution. The API—application program interface—translates the solution to ground vehicle directions. Using WiFi, it is a key communication hub from the aerial to ground vehicle. This is visualized in figure one below.

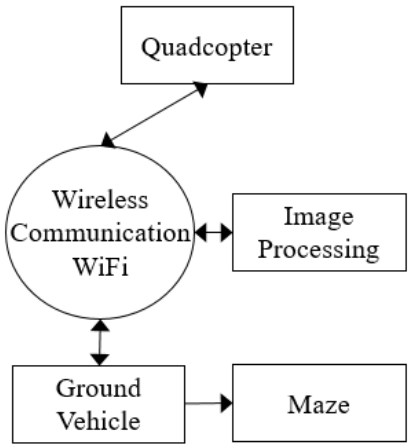

Fig. 1. The major system flowchart shows the integration between the quadcopter, ground vehicle, and image processing.

The ground vehicle was also designed from scratch, aside from a donated chassis. The chassis is sturdy, although unpreferably large at a cubic foot in size. A few key specifications include running for at least fifteen minutes at a speed of at least one sixth meter per second, along with wireless communication up to thirty meters.. Its size allows it to house many components, essentially making it the heart of the project. It features motor controllers, an MSP430, and a Raspberry PI 2 as part of the ground station computer. It also includes DC motors, ultrasonic sensors, a transmitter/receiver, and a power supply system. The embedded control program for vehicle routing along with the motor controller circuits were a major design aspect of this system.

The 3D maze itself contains no technical aspects, although its specifications play a major role in each system's specifications and integration. Its wall's and background are painted bright orange for aerial vehicle object detection from above. The tops of the walls use non-reflective black tape to give enough contrast for the image processing and maze solving. The width and height of the walls must give enough room for the ground vehicle to sense its location and room for the turning radius of the ground vehicle at corners. Size and complexity of the maze are easily modifiable, as maze material is made of fourteen inch long, six inch tall, one inch thick blocks of foam.

III. Ground Vehicle System Components

This multi-system effort is best presented beginning with the ground vehicle components; the heart of the project. Whether purchased or designed, a technical introduction to both hardware components and their software aspects will be discussed before moving to the other systems.

A. Chassis, Motors, and Wheels

The Pololu Dagu Wild Thumper Chassis is large at about a square foot, with a rugged suspension system and all-terrain wheels. This impact's the large scale of the maze. However, it is lightweight and large enough to be seen by the aerial vehicle. It is covered in 4mm holes that make it easy to mount the PCB and other modules. Below the chassis is a compartment to house two sub battery packs, and built in size 34:1 gearboxes house four brushed DC motors. At half power of 7.2 volts, the motors run at up to 350 RPM, or 4mph. They are a two wire connection, without the internal circuits of servos. Speed calibration is regulated using pulse width modulation delivered by the motor controller circuits.

B. Ground Station Computer

The Ground Station Computer (GSC) stands as the technical command and control center for both the ground vehicle and overhead drone alike. The Raspberry PI 2 Model B was chosen for its quad core 900MHz processor and 1GB RAM module. The memory performance plays a significant factor in ensuring how well the system performs overall with all software being ran concurrently. The PI also runs off 5V, is small enough to fit on the ground vehicle chassis, and has UART for communication, meeting our specifications.

The MSP430G2553 embedded processor is a low power system with a voltage range of 1.8V to 3.6V, and 20 available pins, aside from VCC, TST, and GND. Pins 3 and 4 are used to communicate with the GSC via UART. Table 1 further displays the allocated pins of the whole system. The MSP430 is programmed using the Energia IDE which allows for easier integration between non-native Arduino components and the MSP430.

TABLE I. SELECTED MSP430 PIN OUT

MSP Pin #	Short Description	Application
1	VCC	
2		
3	UART Receiving	UART
4	UART Transmitting	UART
5	Echo pin for the sensors	Sensor Interface
6	Select line 1	Sensor Interface
7	Select line 0	Sensor Interface
8	Enable pin for MUX	Sensor Interface
9	Trigger pin for sensors	Sensor Interface
10	Left Side Forward Enable	Motor Controllers
11	Left Side Reverse Enable	Motor Controllers
12	Left Side PWM	Motor Controllers
13	Right Side Forward Enable	Motor Controllers
14	Right Side Reverse Enable	Motor Controllers
15	Right Side PWM	Motor Controllers
16	Reset	
17		
18		
19		
20	Ground	

C. Motor Controllers

Four SN754410 motor controller chips—one for each motor—are connected to the MSP430 embedded processor. The chips can actually handle one motor per 8 pin side, and up to a 36V input. However, the max rated current is 1A along one side. Testing showed that current draw from the motor could be as high as 2.2A, therefore four total chips are utilized, with capacitors between each VCC and ground reduce noise from various parts of the circuit [1]. A single motor controller configuration is displayed in figure two on the next page.

The motor controllers communicate with the MSP430 via simple binary signals that indicate clockwise, counter-clockwise, or no movement. Table II displays the various logic combos. A PWM effect can be applied to the motor through the enable pins, which is useful for course correction. In total, the four motor controllers take up six pins from the motor controller, as allocated in Table I.

TABLE II. LOGIC FOR CONTROLLING MOTOR CONTROLLERS

EN	1A	2A	FUNCTION
H	L	H	Turn Right
H	H	L	Turn Left
H	L	L	Fast Motor Stop
H	H	H	Fast Motor Stop
L	X	X	Fast Motor Stop

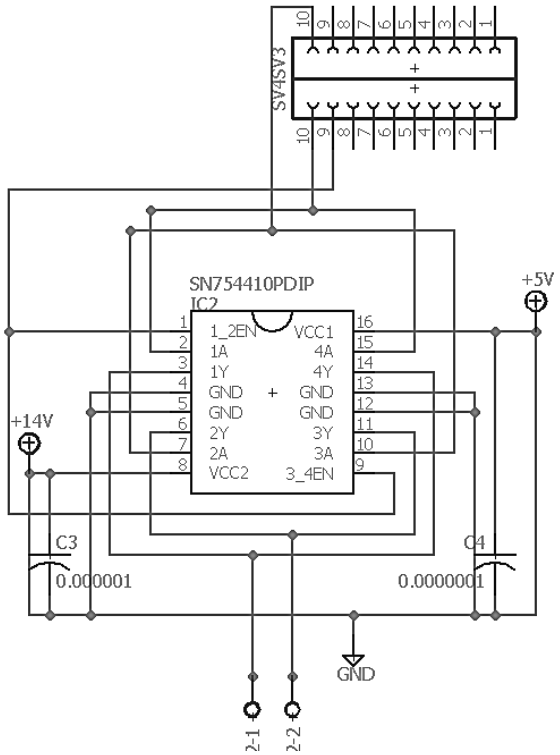

Fig. 2. Single left side motor controller schematic, featuring the SN754410 chip mapped to pins 10, 11, and 12 on the MSP430.

D. Ultrasonic Sensors

To eliminate constant computer vision analysis from the aerial vehicle, three SR-04 ultrasonic sensors were placed on the front and sides of the ground vehicle. Its purpose is to sense the walls of the maze to prevent collision and failure of the project. An ultrasonic sensor was specifically chosen due its sufficient range of five feet, on top of accuracy to the millimeter. Such accuracy is valuable in error correction of drift. As the vehicle traverses the maze, the strength of the motor controllers can be adjusted accordingly.

The sensors run off 5V DC power. They are activated in sequence, since firing in parallel could cause interference. Its only downside is the number of pins required of the MSP430, with a pair of trigger and echo pins for each sensor. To mitigate this, a multiplexer was implemented, which uses a single trigger and echo pin, and three select lines. It only saves on pin but allows for the additional sensor if needed. The MSP430 Pin Out in Table I can be referenced. The CD74HC4052E MUX/DEMUX chip itself runs off the same power the sensors do. Similar to the motor controller circuit, it includes capacitors between each VCC and ground to reduce noise from various parts of the circuit. The sensor interface is displayed in figure 3.

The sensors were mounted on the chassis of the ground vehicle using custom made mounts that pointed the sensors 30 degrees from horizontal towards the ground. This was done to increase the accuracy of the sensors, to ensure contact with the sonic beam and the wall, and to prevent the contact of the beam with the wheels.

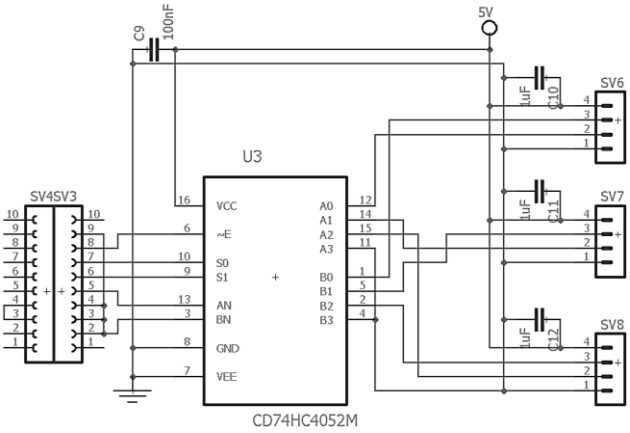

Fig. 3. The sensor interface schematic, from left to right: the MSP430 chip, CD74 MUX/DEMUX chip, and three ultrasonic sensor outputs.

E. Power System

Two 2S lithium polymer batteries—each rated at 7.4 volts—supply 14.8 volts to the whole ground vehicle with a capacity of 3200mA when wired in series. A parallel configuration to double the current draw was considered, but more components depend on a higher voltage. Low voltage alarms monitor the batteries, as going below six volts per battery would permanently damage the cells [2].

The four motor controllers and DC motors receive an unrestricted flow of 14V from the battery and draw 2A. To power the sensors and other sensitive electronics, a two stage voltage stepper design is embedded in the PCB. Both voltage stepper circuits were designed with the help of Webench, so a flowchart overview is displayed in figure 4 instead of the circuits themselves.

The primary chip used in the first circuit is a TPS54340 buck converter. It steps down the voltage from the 14V

battery source and outputs 5V and 2A. This supplies the Raspberry Pi and sensors.

The secondary chip in the next circuit is a TPS54233 buck converter. It takes the previous 5V circuit as an input, to step down to 3.3V and 0.5A. This circuit only supplies power to the MSP430 embedded processor, which doesn't pull many amps in its operation.

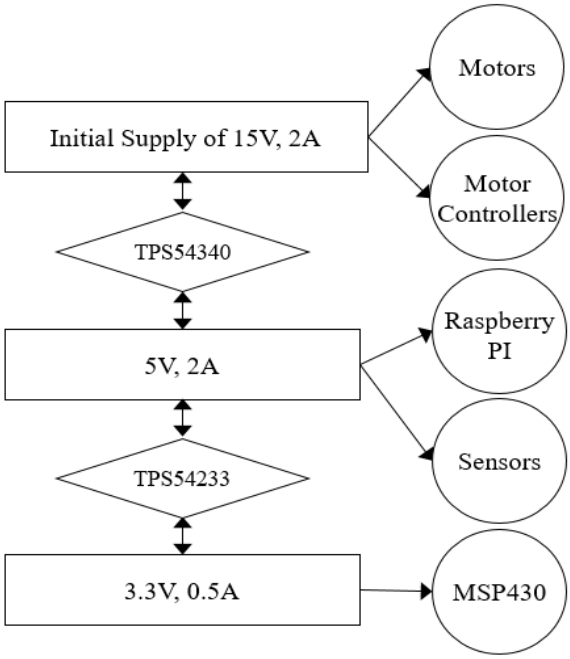

Fig. 4. Ground vehicle two-stage power supply concept, including the voltage stepper chips, outputs, and modules.

IV. Aerial Vehicle System Components

The multi-system effort continues with the Maze Zone Drone itself; the most costly system. Parallels in system components exist between the aerial and ground vehicle, such as the use of lipo batteries and DC motors. Once the aerial vehicle hardware is discussed, the remaining design will lie in software, image processing, and integration.

A. Frame, Motors, and Propellers

The symmetrical X-type frame of the quadcopter is ideal for stability, hovering capability, and efficiency. The flamewheel F450 glass fiber frame was selected, with a total approximate weight of 1300 grams.

Similar to the ground vehicle, four sensorless brushless DC motors were used, although selecting them is more complex. For desired mobility, a rule of thumb is to use motors that generate twice the thrust of the weight of the quadcopter. Equation 1 gives the preferred thrust.

$$\frac{2 \times 1300\ grams}{4\ motors} = 650\ grams\ of\ thrust\ per\ motor \quad (1)$$

Four plastic, "slow fly" propellers are utilized, with proper size and pitch taken into account mostly to maximize efficiency.

B. Flight Controller

Although maneuverability and stability is critical to success, autonomous control is the main focus of this system. The flight controllers will be responsible for and running the open source autopilot, sensing a multitude of measurements, and in turn sending PWM signals to the electronic speed controllers.

The APM 2.6 flight controller specs are not that impressive, with a 16MHz Processor and 256KB flash memory. However, it's enough to get the job done at a very cost effective price. It features an inertial measurement unit—including a 3-axis gyroscope and accelerometer—barometer, receiver input, telemetry port, GPS port, and three communication ports. The external GPS module selected from 3D robotics also has a magnetometer for alignment and is designed specifically for the APM 2.6 [3].

C. Electronic Speed Controllers

The electronic speed controllers are necessary to convert PWM signals from the flight controller into precise control of the brushless motor. Three identical phases control the three phases of the motors: one will be positive, the other negative, and the third to control the timing of the motor. Due to the design costs of four PCB boards and extensive time to fine tune resistance and capacitance values, selecting a commercial electronic speed controller such as the Afro 20A ESC was decided. It is designed to be used with the selected battery and can support the maximum amperage draw of approximately 120% the max draw of the motors. SimonK firmware was designed specifically for use in reflashing this controller

D. Power System

One 3S lithium polymer battery—three cells with a nominal voltage of 3.7 volts—supply 11.1 volts to the whole aerial vehicle with a capacity of 3200mA. The electronic speed controllers and DC motors receive an unrestricted flow of 11V and max draw of 2A. The flight controller, receiver, and camera transmitter require 5V. Luckily, the flight controller contains built-in battery eliminator circuits which can power these components. This effectively eliminates the need for a separate battery or additional buck converter circuit, as the ground vehicle utilizes [2].

IV. Communication

With all the hardware in place, nothing can truly be accomplished without making the systems interact. Two main channels through the air will be used: (1) to transmit and receive telemetry data from the flight controller, and (2) to transmit data from the onboard camera, to the ground vehicle.

A. Transmitter/Receiver

A standard 2.4 GHz four channel transmitter and four channel receiver are needed to manually control the quadcopter. This will include throttle, pitch, roll, and yaw. ArduPilot, as the control station, can switch among various flight modes for autonomous control.

The Tx/Rx communicates with the flight controller through a UART port. On the ground vehicle, the Tx/Rx communicates with the Raspberry Pi through a USB connection.

B. Telemetry

The Mavlink protocol will be implemented to autonomously control the APM 2.6 flight controller. For telemetry, the 3D Robotics telemetry Radio set is compatible with both UART, the Pi, and APM. As specified earlier, it exceeds our minimum connection strength of 50 meters, to over 400 meters. Yet, a broken connection will send the quadcopter back to home coordinates for safety purposes.

C. Image Feed

Due to FCC regulations, the video feed from the camera will be transmitted at 5.8GHz. A high quality, low field of view camera is required. Due to the varying quality of image feed signal over wireless sources on such a low powered system, We have developed several different approaches to getting a clear image, though two methods have emerged as the most viable for this project. The first method is as described, transmit the feed from a from a FPV camera along the 5.8GHz channel to the receiver on the ground vehicle.

The second method involves a but more hardware than the first but produces a far greater image clarity by far which is to mount a separate Raspberry Pi on the quad copter and have it set up a server-client relationship with the Raspberry Pi on the ground. With such a system, a web-camera of much higher quality can be used and the only information that would need to be transferred down to the Raspberry Pi on the ground would be the solution string.

As both systems are viable, which we use for a test depends on the weather and lighting conditions of the test area and the complexity of the quadcopter's search pattern.

V. Software Detail

A majority of the project is software based. Many small subprograms make up the main flow of information from quadcopter, to ground station computer, to embedded processor. C, C++, and Python are the main languages used.

A. Ground Vehicle Embedded Main Program

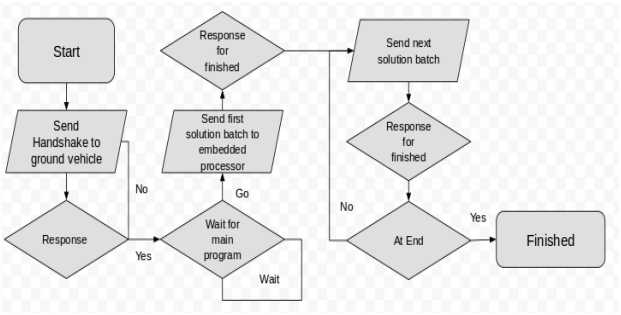

Fig. 5. The process for gaining the solution of the maze

The embedded main program drives the motors and allows the ground vehicle to feel its way through the maze using the ultra-sonic sensors. The embedded main program was written in C using the energia IDE for its ease of use and pin specific libraries. This main program keeps track of the maze turns that it has made as well as future turns to make. The motors that are connected to the motor controllers that are controlled by our embedded processor propel the vehicle along at a pace that could not be controlled so a PWM (Pulse Width Modulation) component was added to keep the speed of the vehicle in check.

The main microcontroller also is required to communicate to the Raspberry Pi that is mounted to the ground vehicle, so UART communication out via the pins was required. Energia had many libraries that facilitated the communication via the Serial.print() commands that were later combined to create entire UART functions to oversee the transfer.

B. Embedded Processor Sensor Software

The inspiration for the sensor software came from a tutorial on the SR-04 sensor found on youtube [3]. The main part of the program was reworked to change the program from one that returned distances in inches to one that returned distances in centimeters for accuracy as well as removing the bits that transformed the distance into an audio frequency. The activation of the sensors through the software is staggered to prevent interference between the sensors which would hinder the vehicle's progress.

The software also reduces the effective range of the sensors, which was intentional, as the maximum range of the sensors, being 15ft or so, was enough of a delay roundtrip at the speed of sound to cause a noticeable and debilitating delay in the program. To work around this, the cut off timer on the sensors was reduced down to .0058 seconds, or the time it takes the beam to travel one meter. With this change, the program speed quadrupled and became viable.

C. Ground Vehicle CPU Main program

The CPU main program is comprised of several smaller programs and mainly serves as a link to all the separate programs and is written in Python for simplicity and versatility on the Raspbian OS. The greatest challenge with this code was creating compatibility between all the separate functions and routines, some of which were written in C++. The general flow of the main program is as follows and can be seen visually in figure X, the program listens for valid communication handshakes from the embedded processor as well as the quadcopter through the interface for mavproxy. The program then enters standby until the user gives the signal to begin, once given the main computer gives instructions to the drone using the mavproxy terminal. Then, as the drone is executing its grid-search pattern for the maze, the Raspberry pi is analyzing the data in the form of images that are streamed from the mounted camera. Once the maze has been found, solved, and its solution extracted from the image, the main program then feeds that solution set to the ground vehicle for it to then solve the maze.

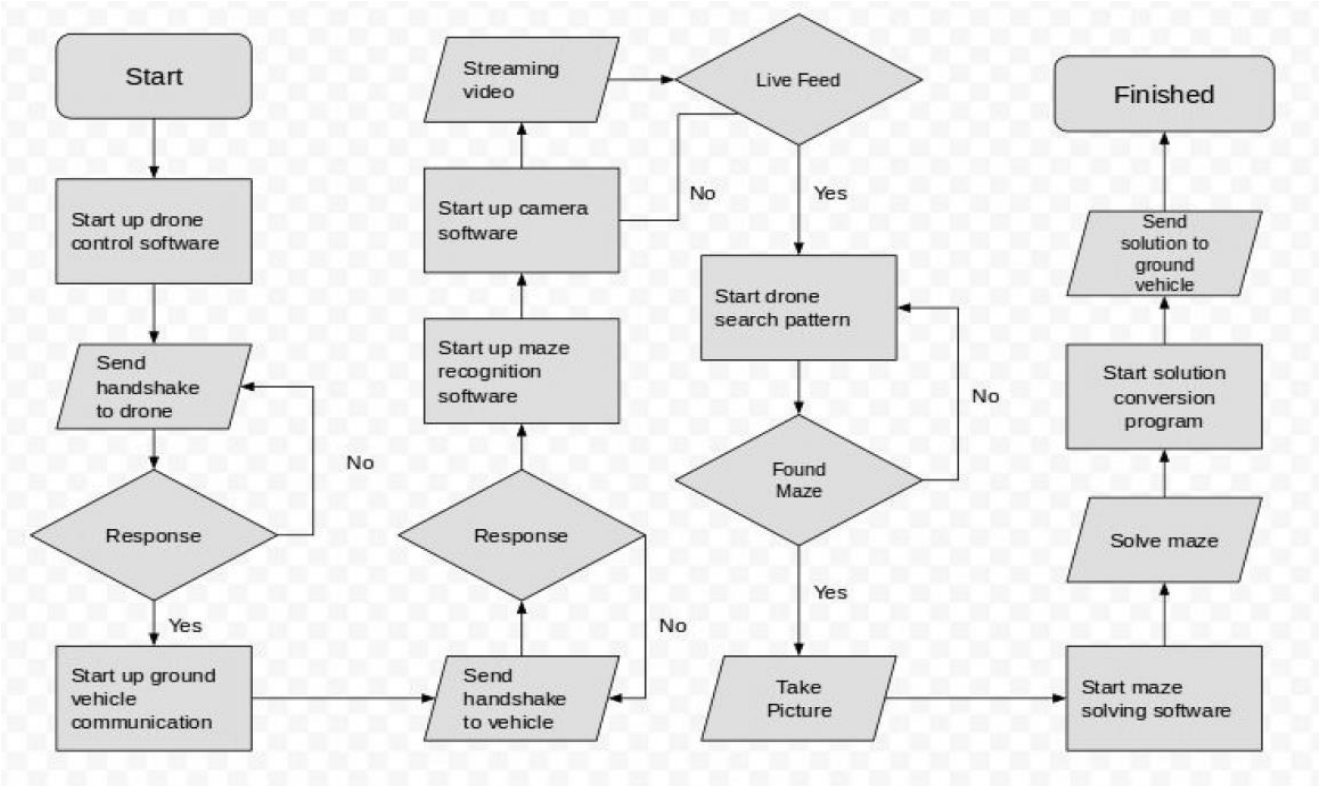

Fig. 6. Shows the main program flow, made up of many smaller programs and different languages

D. Maze Solution Translation

The maze solving sub program was built to translate the image solution of the maze, into the string solution that was necessary for the embedded processor. The translator works by first finding the start and finish points of the maze as marked by a yellow and blue square. Once those points are located, the program starts at the beginning and analyzes every pixel in the four directions around it, with a 30 pixel range in every direction. The numbers that are returned indicate the number of red pixels, white pixels there were between that pixel and 30 pixels away, as well as what pixel color it ended on. With this system then comparing the distance as well as the direction of greatest concentration of red solution pixels, soon every turn from the start to the end is mapped out and saved as a string to be transmitted to the ground vehicle.

V. Image Processing

The image processing portion of this project is accomplished using a blend of commercial CV software as well as some solutions necessary for this specific project. This is accomplished in two parts software wise, and both systems are relatively independent of one another. The first system is the maze locating program which searches for the unique maze background color against the testing surface. Once this images is found it is then analyzed for which place in the frame it is in and those results relayed to the drone to adjust altitude and position. Once the maze has been centered in the frame and the image is clear, the maze solving portion of the program solves the maze morphologically using openCV calls. This solution image is then analyzed to get the turns along the solution path and the results are forwarded to the embedded processor

A. Color Filtering

Color coded items—the maze and its waypoints—need to be easily distinguished from its surrounding, which is why the choice of neon colors (orange, blue, green, white, and black) are utilized. Due to lighting, using the RGB color system is not practical. Instead HSV values are returned.

The color filtration program was necessary for this project, as we use thresholding to gain the HSV values of the image we have taken to more accurately decode the values for the openCV software to work with. HSV stand for Hue, Saturation, and Value and with those three values you can determine an object's true color value regardless of the lighting level and presence of shadows, though shadows do cause problems with the way the CV software works.

B. Object Detection

Object detection is a major part of the autonomous drone and image capturing process. The object detection software relies heavily on the work of the color filtering program as it splits each picture it receives into several quadrants and looks for the specific value that is our maze background color, inside said quadrants. When when a quadrant is labeled as containing an object with a similar HSV to the target, it reports back to the main program that an object has been detected in the specific quadrant. Once the main program has the full analysis of the entire picture it can then work out where the maze is in relation to the quadcopter and move the quadcopter to get a better image by aligning it into the center of the image. Once the image is in the center of the frame, the quad copter enters high precision hover mode to take the final image for processing.

C. Edge Detection

The edge detection software utilizes morphological transformation of the maze to arrive at a solution. Due to this type of transformation to arrive at a solution, there are several constraints to the maze. Primarily the maze must have exactly one beginning and one end, meaning there cannot be multiple exits or starting points for the maze. The maze must also have exactly one solution, meaning there cannot be more than on path that leads to the exit from the entrance. Unlike A* algorithm which utilizes a breath first search to discover the optimal solution, being the shortest solution in a solution space of many different paths, this algorithm essentially finds one half of the maze through edge detection since the constraints of the maze determine that there will be exactly two halves to the maze and then subtracts it from the second half of the maze. What remains then becomes the solution to the maze and is indicated by a red path in the image of a width of our choosing.

D. Maze Solving Algorithm

The morphological way of solving the maze is the cleanest and easiest way to solve the maze in minimum steps. It utilizes the power of image processing using openCV libraries. The only drawback of this method is that it only works for a simple perfect maze that has only one solution. The output image from the edge detection

part will be used as an input for the maze solving program. After filtering the edges and separating those from its background, the output will be converted to a gray value

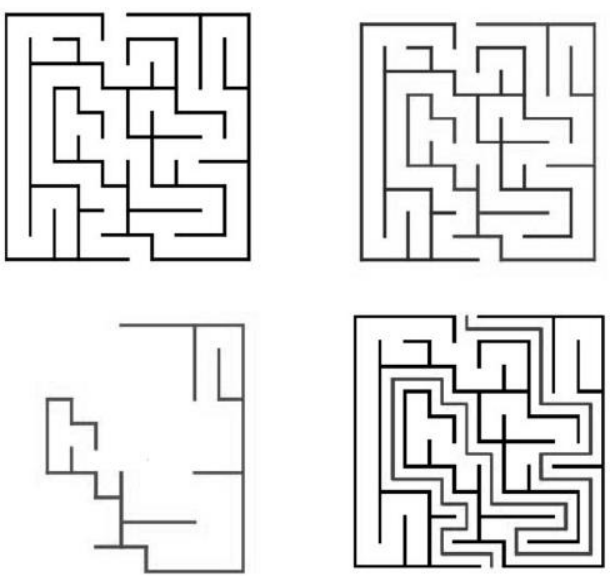

Fig. 7. The process for gaining the solution of the maze

image. Maze solver uses the binary image and solves it using a mathematical morphology. It first labels the walls in the binary image, and then separate the walls in the image and splits the maze in two different parts. The series of instructions to translate the maze solution was described in the software detail.

VII. Conclusion

Through completing this project, we have gained valuable experience in the finer points of both software development and hardware development to accomplish a specific goal. All of us came into this project with little practical experience in designing a board, using the tools available to us or utilizing tools offered by companies, but we now leave with the knowledge to complete a full range of tasks. With the help of our education, and the limited resources available to our project and in spite of severe setbacks with regards to building the PCB, we have managed to create a functioning and successful project that will inspire future generations of engineering students to succeed.

The Engineers

Ehsan Falaki is a 31 year-old graduating Computer Engineering student who plans on graduating in May of 2016. He plans to continue his education as a grad student majoring in computer science at Georgia Institute of Technology as well as working as a full time software developer.

Tanner Foster is a 22 year-old graduating Computer Engineering who plans on graduating in May of 2016. After graduation he plans on working as a software developer in Orlando or Tampa.

Matt Szewczyk is a 22 year-old graduating Electrical Engineering student who is seeking a job within Lockheed Martin in Orlando, FL, as a reliability engineer in product support.

Justin Yuen is a 22 year-old graduating Electrical Engineering student who plans on graduating in May of 2016. He currently works as an EE intern at Athena Technologies, LLC. He plans to follow radio communication networking throughout his career

Acknowledgement

The authors wish to acknowledge the assistance and support of Dr. Samuel Richie and Dr. Lei Wei; University of Central Florida. We would like to thank Soartech, Boeing/Leidos, and Michelle Foster Photography for funding to make this project a reality.

References

[1] *SN754410 Quadruple HalfH Driver* . Dallas: Texas Instruments Incoporated, Jan. 2015. PDF.

[2] Lott, Wes. *How to Choose a LiPo Battery* . Digital image. *RCPowers* . N.p., 20 Jan. 2012. Web. 6 Dec. 2015. <www.rcpowers.com/community/threads/howtochoosealipobatterytutorial.10855/>.

[3] 7] "Dimension Engineering BECs for Every R/C Plane and Helicopter Frequently Asked Questions." *Dimension Engineering BECs for Every R/C Plane and Helicopter Frequently Asked Questions* . Dimension Engineering, n.d. Web.06Dec.2015