LIGHTER-THAN-AIR ROBOT SWARM

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted by

Alec Newport (acn55), Junghsien Wei (jw2597), Pooja Menon (pjm355),

Priyanka Dilip (ped48)

MEng Field Advisors: Joe Skovira, Kirstin Petersen

MEng Outside Advisor: Nialah Wilson

Degree Date: May 2020 (acn55, jw2597, pjm355)

Table of Contents

Table of Contents (continued)

Abstract

Masters of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Lighter-than-Air Robot Swarm

Authors: Alec Newport, Junghsien Wei, Pooja Menon, Priyanka Dilip

Abstract:

The goal of this project is to develop the initial platform for a lightweight, autonomous drone that can be deployed in a swarm to provide navigation assistance to humans attempting to escape a building during an emergency. Each drone is designed to traverse a hallway and project an indicator on the path to guide stranded people to the nearest exit safely. The prototype is currently capable of manually-controlled flight. It uses a Raspberry Pi Zero W for computation, AprilTags detected through a Raspberry Pi camera for localization, and time-of-flight sensors for obstacle avoidance. This sensor data is sent over a WiFi connection to a base station program running on a PC, where a human operator can view it and send flight commands back to the drone.

Alec Newport's Executive Summary

My initial work focused on testing and evaluating ultrasonic distance sensors for localization and obstacle avoidance. I investigated using short range sensors on the sides of the robot to ensure it was maintaining a sufficient distance from walls as it navigated. I also tested using long range sensors on the front and bottom to determine its approximate position in the hallway and its altitude. These sensors proved to be too noisy, unreliable, and heavy to be of any real use, so they were abandoned. I then moved to attempting to install the Robot Operating System (ROS) on the Raspberry Pi Zero. After some struggles getting it to install correctly, I learned that the Pi Zero does not have the capability of running ROS. By this time, we had determined that we would not be using a Vicon system for navigation, which had been one of our primary reasons for looking into ROS, so this was abandoned as well. The remainder of my time was spent on the base station software. I wrote the user interface in Python using the PyGame library to run on a PC. This program communicates using the Python Socket library over WiFi with a server running on the robot. The software can be used by a human operator to view data coming from the robot, as well as to send it drive commands. The interface allows for three different control methods: fully-automatic, in which the robot is completely in control and chooses its own waypoints; waypoint control, in which the user sends waypoints to the robot, and the robot determines how to navigate to them; and fully manual, in which the user inputs precisely how they wish the robot to move. The latter is the only one we have working currently, but we anticipate future work on this system will increase its autonomy. As work is done in these areas, we believe minimal changes will need to take place to the base station to allow for this.

Junghsien Wei's Executive Summary

In the first semester, I used Solidworks to design the frame of the blimp and studied for the projector principle and I tested several light bulbs and flashlights for projection, so I used the convex lens that we ordered and followed the reference I searched, and the image focused well. I used Solidworks again to create a projector case with a 3D printer and laser cut. Moreover, I studied the principle of the dc motor and wrote the motor program that I learned from ECE5725. I wrote several programs for testing each motor and expanded them to the complete running program. Priyanka and I soldered several parts on our first prototype and measured the maximum lifting weight of the balloon in order to test the entire system on the balloon. We had several tests with different types of balloons and bought many helium tanks from the Party City. I also studied the LED circuit for the connection between the GPIO and the projector with Dr. Joe, and I implemented the blink code to the motor program. Last, I tried to apply the motor program with the base station program, but I got some error messages, so I asked Alec for help. We debugged the program and finally the code worked well in the demonstration.

In the second semester, I designed a PCB for our second prototype in order to improve its weight and performance. I also modified the flight-sensor program, Apriltags detection program and the server control program of the system. Then, I tested the program performance of the system and assembled all the parts on the PCB. Besides, I re-designed the new projector and the motor chassis for the second prototype.

Pooja Menon’s Executive Summary

In the first semester, my contribution to the project was collecting information about the navigation of the blimp using ROS and trying to integrate it on the Raspberry Pi zero. I also worked with my teammates and assisted them in flight testing and assembly and testing of the vicon tracking on the blimp.

In the second semester, my contribution to the project was working with my teammates and developing a customized blimp envelope that is designed to lift the weight of the components used and still remain compact.

Acknowledgement

- I used this installation guide as reference for ROS configuration ROSberryPi/Installing ROS Kinetic on the Raspberry Pi - ROS
- Getting started with ROS **h**ttps://github.com/SamSpaulding/ros_raspberry_pi_zero
- I used this to get a basic understanding of the vicon tracking system Vicon Motion Capture Basic Explanation
- I used to to learn about vicon integration with ROS http://wiki.ros.org/vicon_bridge
- I used this to make the mylar balloon envelope How to Make your own mylar balloon
- I used this for understanding different blimp envelope designs Spacial: Can Startup Compete With DJI As The Go-To Media Drone

Implementation Details

- The ROS was to be used for navigation initially. The vicon tracker would track the movement of the blimp and send the coordinates to the blimp as well as the base station.
- This information would then be used to navigate around the hallways
- The individual components were causing weight issues and the need for a bigger envelope which would defeat the purpose of a lightweight drone.
- Thus making our own blimp envelope design would make the process much more easier as we could customize the blimp as per our requirements

Priyanka Dilip's Executive Summary

In the first semester, I worked on and contributed to all hardware/mechanics aspects of the project: blimp envelope design, circuitry implementation, flight testing, projection system.

In the second semester, I was involved in nearly all aspects of the blimp, from electronics/mechanics to software implementation. In the first half, I was focused on weight optimizations (both in terms of envelope design and electronic hardware), as well as the AprilTag camera detection implementation. Although we had made great strides in cutting down payload weight, the additional functionalities that we wanted to build this semester (sensors, camera etc.) required a sizable volume of helium to create sufficient lift sans motors. I went through several iterations for the envelope: from previous semester's increasingly larger and more fragile balloons, to a single monolith consisting of several smaller balloons enclosed in a mylar sheet, to attempts at crafting a custom envelope from mylar sheets. Finally, I settled on two 3ft balloons which I sewed around the seam to create a rigid, aerodynamically stable structure.

I studied and drafted an implementation for the AprilTag detection and navigation methodology, as well as PCB design and printing in collaboration with the MakerSpace.

In the second half, due to quarantine distance restrictions, I focused more on the software. I wrote drafts for the integration code that would orchestrate all modules of the project together, building on previous code by Alec. I also tested usage of the time-of-flight sensors and how to distinctly connect them via a single GPIO pin. My work on drafting the software implementation of each of these hardware modules was directly to support the final code and testing written and conducted by Sam. Finally, I documented progress as Weekly Report sections submitted to the Advisors.

1 Introduction

Natural disasters and other emergencies often require large buildings to be evacuated quickly. However, these situations provide numerous obstacles. Vision may be obscured, people may panic, and dangerous conditions may prevent rescue workers from reaching people trapped inside. In such situations a robot can provide assistance by locating the areas where people are stuck and providing guidance to safely navigate out of the building.

The main objective of our project is to develop a lighter-than-air, autonomous drone that is versatile enough to accomplish these and other various tasks while remaining simple enough to be scaled up to a swarm. The primary goal for the first semester was to construct the first drone prototype. We focused on testing various sensors while reducing weight wherever possible. In the second semester, we implemented advanced functionality such as a camera and distance sensors. These give the robot the capability to autonomously find its way to the exit and navigate complex environments. In addition, we completed a projection system that projects an arrow visible on the ground indicating the exit's location. Finally, we completed a base station with which the blimp can wirelessly communicate to display information and receive commands as necessary.

2 System Design

2.1 System Overview

The major components of the system are as follows:

- Central computation on a Raspberry Pi
- Lift generated using helium-filled balloons and propulsion using propellers
- Navigation using a camera and AprilTags
- Obstacle avoidance using time of flight sensors
- A system to project an arrow on the ground
- A base station program running on a PC that communicates wirelessly with the robot

An overview of the system is shown in Figure 2.1.

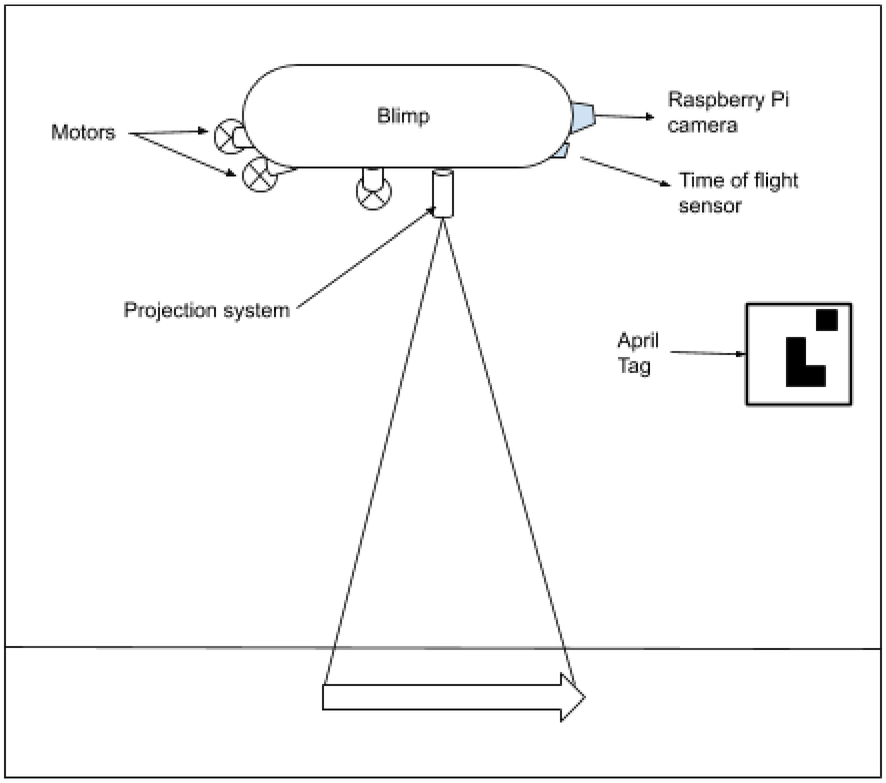


Fig 2.1 High level system design

2.2 Hardware System Design

A Raspberry Pi Zero W[1] is used as the robot's central microcontroller. There are three motors attached to the system. The two rear-facing motors drive the blimp forward and backward and turn it left or right. The one at the bottom controls the blimp's altitude. The motors are controlled with a H-bridge, and the Raspberry Pi camera[2] is used to navigate. There is an additional booster circuit needed to provide the required power to the whole system. The block diagram for the robot's hardware is shown in Figure 2.2.1.

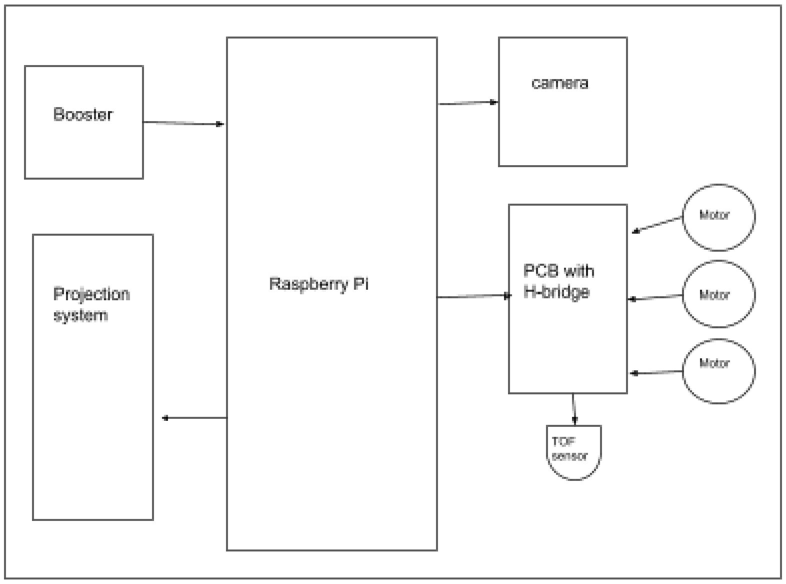


Figure 2.2.1 Hardware system block diagram on the blimp

2.2.1 Raspberry Pi Zero W

The central component of the hardware design is the Raspberry Pi Zero W, seen in Figure 2.2.2. We chose this for its light weight, only 9.8 grams, and its ease of use. It serves as the central microcontroller and is used to control the motors and time of flight sensors, communicate with the base station, and navigate by analysing the images from the camera to search for AprilTags using computer vision.

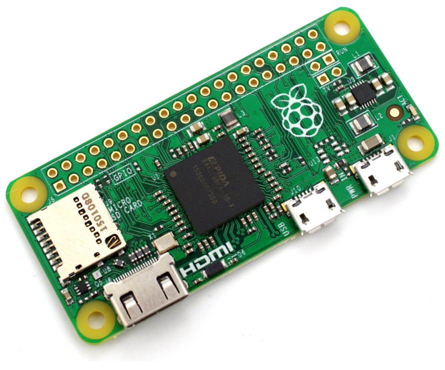


Figure 2.2.2 Raspberry Pi Zero W

2.2.2 Power Booster

In the first semester, we used an Adafruit PowerBoost 500 Basic voltage booster module[3] to provide power to the system. This module is a DC/DC converter that can draw power from a 1.8V battery and convert it to 5.2V that is the required voltage for the Raspberry Pi. The maximum output current is 500 mA, with 700KHz high-frequency operation. In the second semester, we replaced the Adafruit chip with an MC34063[4], an inverting regulator to design our own power booster. The chip can supply output switch current to 1.5 A, which allows us to integrate additional components such as flight-sensor or camera on PCB.

2.2.3 Time-of-Flight Sensors

The time-of-flight sensors (TOFs), VL53L0X[5], are used to get estimates of the distances of obstacles from the blimp, up to a range of 1 meter. They estimate distances by sending out pulses of infrared light and measuring the time delay until a reflected pulse is detected. We use three of these sensors: one on each of the left, right, and front of the blimp to detect and avoid obstacles.

2.2.4 Motor Driver

As mentioned in previous sections, the blimp is driven using three DC motors. We want to run these motors forwards and backwards using the Raspberry Pi's GPIO pins. To achieve this functionality, supply the necessary current, and protect the pins from inductive kickback from the motors, we use two Adafruit DRV8833 DC/Stepper Motor Driver[6] chips. Each module consists of two h-bridge circuits, giving them each the ability to control two DC motors bi-directionally. In the second semester, we replaced the Adafruit chip with an L293D[7], which contains quadruple high current half-H drivers. This allows us to integrate the component on the PCB.

2.2.5 Raspberry Pi Camera

The Raspberry Pi camera is used in conjunction with AprilTags[8] in the environment for localization and navigation. The microcontroller is programmed using OpenCV[9] to detect and analyze these tags, and the corresponding information is sent to the base station. The frame rate for video capture is set to 60 fps.

2.2.6 Projection System

The projection system is made using a 1W LED from Chanzon company[10] and four 10 mm optical lenses[11]. The assembled instruction of the projector is shown in Figure 2.2.3. A small lens is placed in front of the light source with the image film to gather all the light and create a beam of parallel rays that pass through the film. Then, the image is placed between the first and the second focal point of the second convex lens, which will generate an enlarged image. The best length (0.8 cm) between these two focal points is recorded to design the projector case which is then 3D printed into a compact model case.

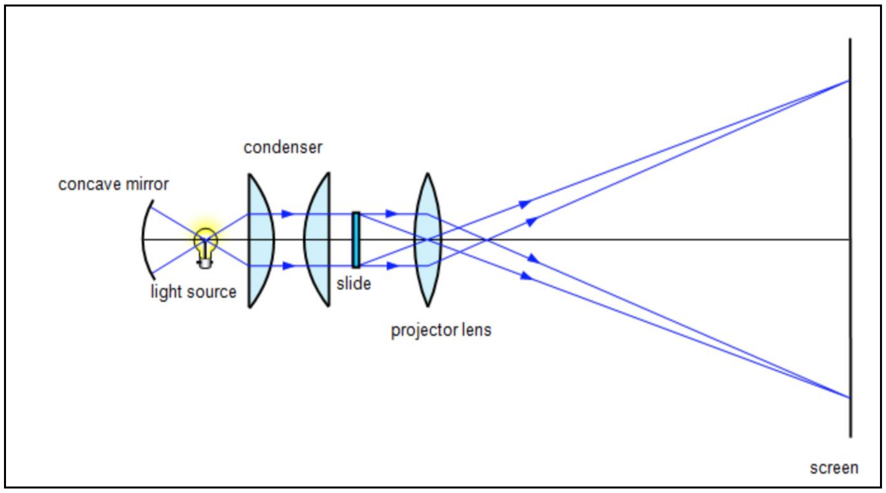


Fig 2.2.3 Projection principle and assembly instruction [12]

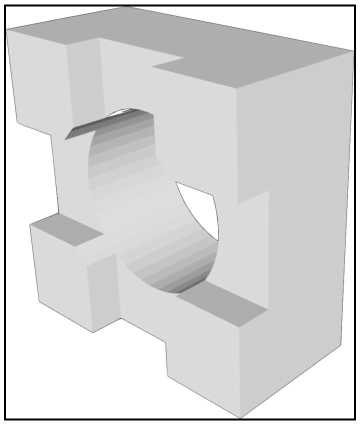


Figure 2.2.4 Projector case

2.3 Base Station Design

The base station serves as the link between the robot and human operator. Its code consists of two parts: the client, which runs a PyGame[13] GUI on a PC, and the server, which is running as part of the code on the Pi. The two communicate over WiFi using Python's built-in Socket library[14]. The client sends a request or command to the server, and, if required, the server returns a response to the client. Both messages are sent as encoded strings. A table of these messages is shown in Figure 2.3.1.

2.3.1 Base Station Client

The base station client, whose code is in base_station.py, is the user interface for the system. It is where all of the sensor data is displayed and where the user can enter drive commands to send to the robot. See Figures 2.3.2-2.3.4 for images of the interface.

The table labelled "AprilTag Robot Localization" displays information gained from sensing AprilTags. If a tag is currently visible in the Raspberry Pi camera's field of view, information about that tag will be displayed here. The first column contains the tag's identification number. The second displays the tag's global x and y coordinates in the environment, giving the user (and the robot, if it is making autonomous decisions) an idea of the robot's location. The third shows the robot's distance from the tag in the same units as the x and y coordinates. We are assuming the tags will be placed on the ground, so this number approximates the robot's altitude, or global z coordinate. Finally, the fourth column contains the robot's approximate global orientation in the environment. We are assuming all tags will be oriented at 0° relative to the global coordinate system, so this number is acquired from negating the apparent orientation of the tag relative to the robot.

The second table, labelled "Time of Flight Distances" displays the data received from the three time-of-flight obstacle detection sensors. Each column is labelled with the direction the corresponding sensor is facing. As noted in Figure 2.3.1, the server is also sending status numbers for each sensor. This information is displayed as a different highlighting color. A green highlight indicates that the robot does not think it is in any danger of hitting an obstacle on that side. A yellow highlight indicates it is getting close to an obstacle and care should be taken if moving in that

Request/Command	Request String	Response String
Client and server shutdown command	“quit”	“quitting”
Move forward command	“forward”	None
Move backward command	“backward”	None
Turn right command	“right”	None
Turn left command	“left”	None
Increase altitude command	“up”	None
Decrease altitude command	“down”	None
Stop motors command	“stop”	None
Request current waypoint	“curr wp”	“<wp_x>,<wp_y>,<wp_z>,<wp_θ>” wp_x: Current waypoint global x coordinate wp_y: Current waypoint global y coordinate wp_z: Current waypoint global z coordinate wp_θ: Current waypoint global heading
Set drive mode command	“mode <num>” num: operation mode “0” - full auto “1” - waypoint “2” - full manual	None
Set waypoint command	“wp <wp_x>,<wp_y>,<wp_z>,<wp_θ>” wp_x: New waypoint global x coordinate wp_y: New waypoint global y coordinate wp_z: New waypoint global z coordinate wp_θ: New waypoint global heading	None
Request AprilTag data	“at”	If no AprilTag currently visible: “False” If AprilTag currently visible: “True,<id>,<x>,<y>,<dist>,<ang>” id: AprilTag ID number x: AprilTag global x coordinate y: AprilTag global y coordinate dist: Distance from camera to AprilTag ang: Apparent angle of AprilTag
Request time of flight data	“tof”	“<dL>,<dC>,<dR>,<sL>,<sC>,<sR>” dL: Left sensor distance reading dC: Center sensor distance reading dR: Right sensor distance reading sL: Left sensor collision status sC: Center sensor collision status sR: Right sensor collision status “0” - OK “1” - Danger “2” - Collision imminent
Unrecognized command	“<other_str>”	“Not recognized”

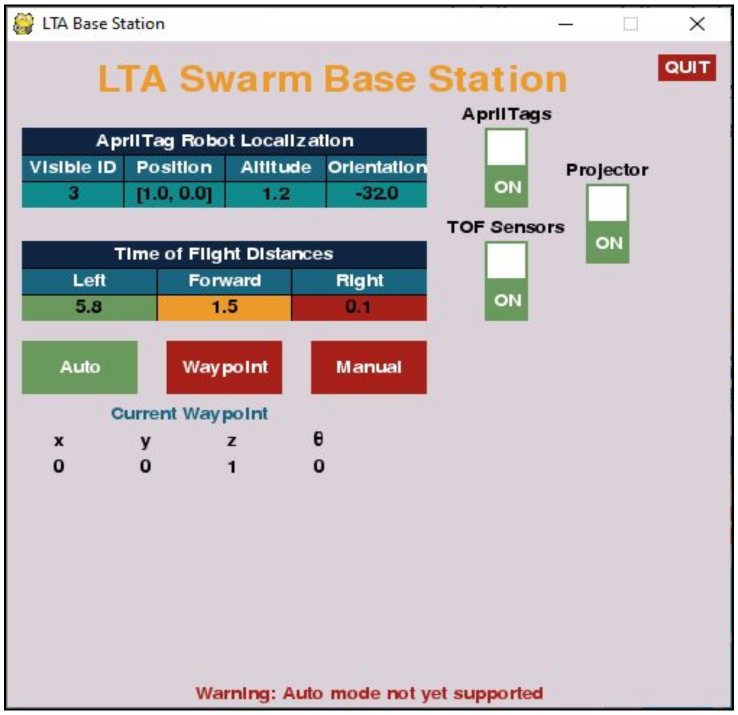
Figure 2.3.1 Base station socket connection strings

direction. Finally, a red highlight indicates that a collision is imminent or has already occured. In this case, if not already doing so, the operator should switch to manual control to move the robot away from the obstacle in that direction to prevent damage.

The three switches to the right of the tables labelled “AprilTags,” “TOF Sensors,” and “Projector” are Enable switches for the corresponding systems. If the “AprilTags” or “TOF Sensors” switches are turned off, the client will stop requesting the corresponding data from the server. The “Projector” switch, on the other hand, directly controls the projection LED on the robot. Turning it on or off will turn the indicator projection on or off, allowing the operator to prevent confusion if the robot is moving to a new location rather than indicating the direction of an exit while operating.

The three red and green buttons below the tables allow the user to switch between drive modes. The green button is the currently enabled mode. The interface currently allows for three modes: fully-automatic, waypoint drive, and fully-manual.

The first drive mode, fully-automatic, is shown in Figure 2.3.2. In this mode, the robot is completely responsible for driving itself. It chooses its own goal orientation, plots its own path to get there, and autonomously follows that path. The current waypoint along the path to which the robot is driving is displayed on the interface using its x, y, z, and θ coordinates.



AprilTag Robot Localization			
Visible ID	Position	Altitude	Orientation
3	[1.0, 0.0]	1.2	-32.0

Time of Flight Distances		
Left	Forward	Right
5.8	1.5	0.1

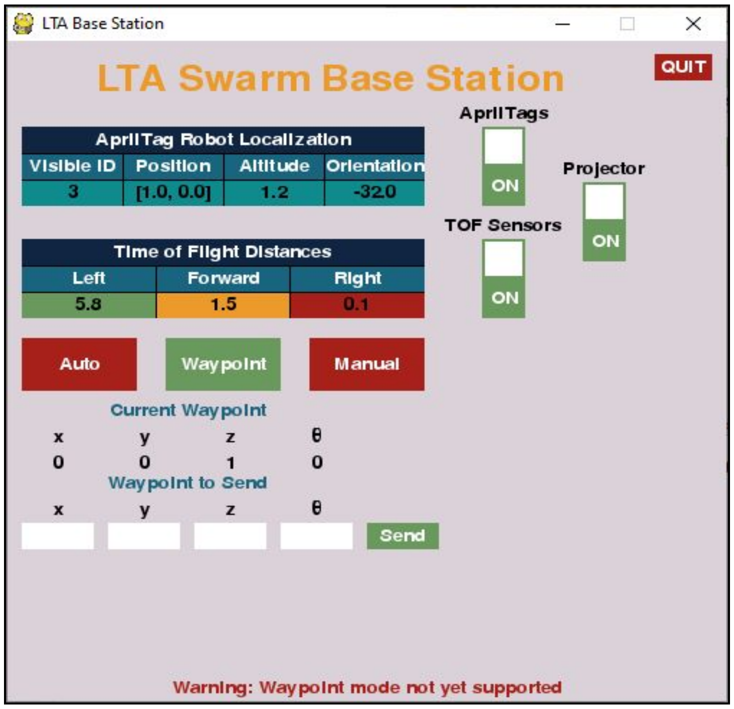
Current Waypoint

x	y	z	θ
0	0	1	0

Warning: Auto mode not yet supported

Figure 2.3.2 Base station client auto mode

The second drive mode is waypoint drive, shown in Figure 2.3.3. Here, the user enters a waypoint to which the robot should drive itself, but the robot autonomously plans and executes a path to that point. Like full-auto mode, the robot's current waypoint is displayed. In addition, text boxes for entering the global coordinates of the next desired waypoint are displayed. The user enters the coordinates and presses the green "Send" button to transmit them to the robot. If any of the coordinates are invalid, a warning is displayed, and the corresponding textbox is activated.



AprilTag Robot Localization			
Visible ID	Position	Altitude	Orientation
3	[1.0, 0.0]	1.2	-32.0

Time of Flight Distances		
Left	Forward	Right
5.8	1.5	0.1

Current Waypoint

x	y	z	θ
0	0	1	0

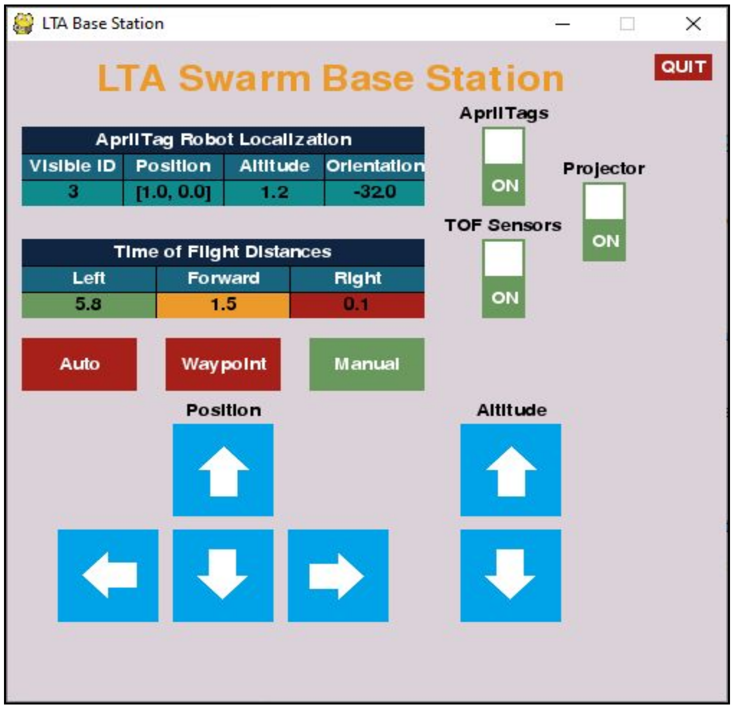
Waypoint to Send

x	y	z	θ

Warning: Waypoint mode not yet supported

Figure 2.3.3 Base station client waypoint mode

The final drive mode is fully-manual, shown in Figure 2.3.4. In this mode, the user is in full control of all of the robot's movements. The left set of arrow keys allow the user to move the robot forward or backward or to turn the robot left or right. The right set of keys allow the user to control the robot's altitude. Pressing and holding any of these keys will drive the corresponding motors, and releasing will stop them. This is currently the only drive mode implemented on the robot, which is why the other two display warning messages at the bottom of the interface. The other modes were left in for compatibility with future versions of the robot.



AprilTag Robot Localization

Visible ID	Position	Altitude	Orientation
3	[1.0, 0.0]	1.2	-32.0

Time of Flight Distances

Left	Forward	Right
5.8	1.5	0.1

Figure 2.3.4 Base station client manual mode

When finished running the robot, pressing the red “QUIT” button in the upper right will terminate the robot’s execution, close the connection, and exit the GUI.

2.3.2 Base Station Server

The base station server code is integrated with the rest of the code running on the robot. When it receives a request or command, it parses the string and gathers the required data or sends the provided command to the robot. For implementation details, see section 3.3.

3 Implementation

3.1 Hardware Implementation

In the first prototype, our primary goal is to make the blimp fly. There are three main components, which are power booster, motor driver, and the raspberry pi zero. The power booster can be used with just the microUSB charge input, battery plug and power outputs. Therefore, we only need to plug in the 3.7 battery to the chip and connect the micro USB to operate the raspberry pi. There is a

5V supply on the power booster which we use to power the motor drive. The motor driver contains two full H-bridges which allows us to drive two DC motors bi-directionally. However, we have three motors in our system, so we need two motor drivers. We then connect the GPIO pin from the raspberry pi to the driver in order to use PWM to control motor speed and direction, which is shown in Figure 3.1.1.

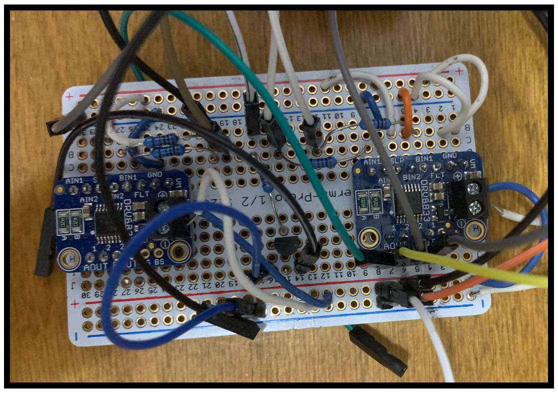


Figure 3.1.1. Motor driver circuit.

We use Solidworks to design the motor frame and the projector case for 3D printing and laser-cutting, which is shown in Figures 3.1.2. The frame design was based on that of the MicroBlimp since we knew that system had worked well in the past. Then, we assemble three motors on the frame by using hot glue.

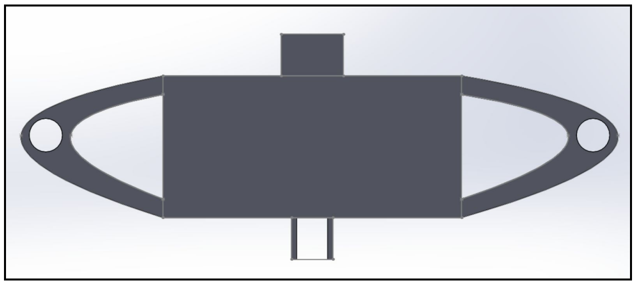


Figure 3.1.2. Blimp frame design.

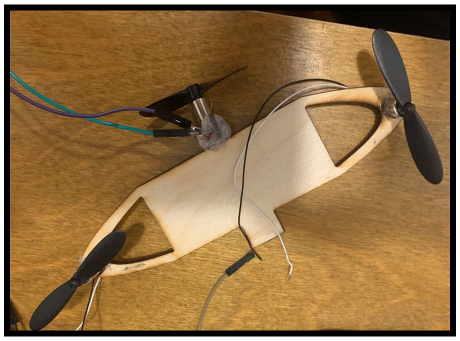


Figure 3.1.3. Blimp structure with motors.

The first prototype works properly, but the entire weight is too heavy, which requires lots of helium for the balloon. Therefore, we want to optimize our system circuit by integrating components to the PCB. We draw the schematic of the entire system, which is shown in Figure 3.1.4. In the second prototype, we have three additional flight-sensors and a transistor circuit for the LED[9]. In addition, we replace the motor driver and power booster with IC chips. We use MC34063[4], a popular switching regulator, to boost the 3.7 V to 5 V. The datasheet for the regulator contains the complete formula chart, shown in Figure 3.1.5, to calculate the desired values as per our requirement. The components' values are shown in the schematics.

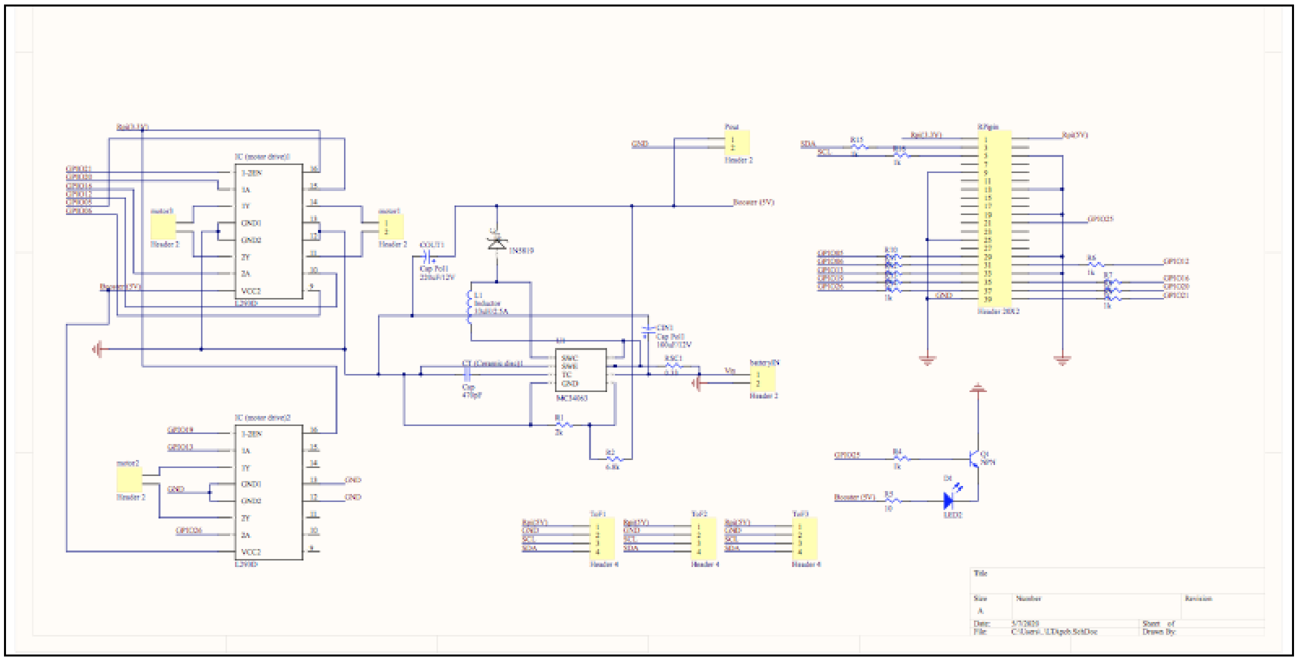
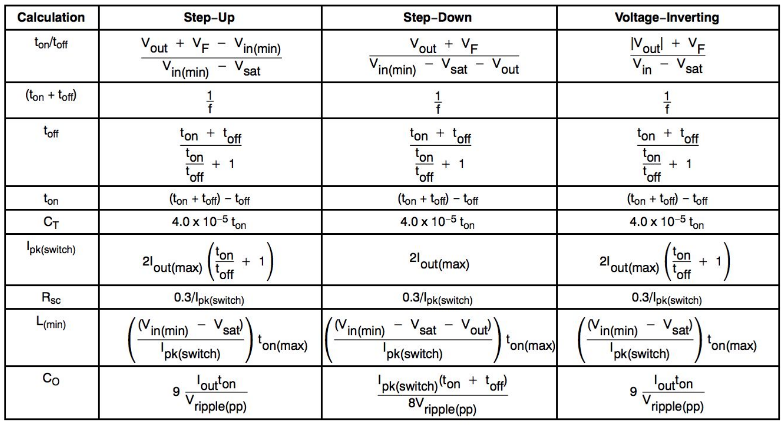


Figure 3.1.4. System circuit schematics.



Calculation	Step-Up	Step-Down	Voltage-Inverting
t_{on}/t_{off}	$\frac{V_{out} + V_F - V_{in(min)}}{V_{in(min)} - V_{sat}}$	$\frac{V_{out} + V_F}{V_{in(min)} - V_{sat} - V_{out}}$	$\frac{\lvert V_{out}\rvert + V_F}{V_{in} - V_{sat}}$
$(t_{on} + t_{off})$	$\frac{1}{f}$	$\frac{1}{f}$	$\frac{1}{f}$
t_{off}	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$	$\frac{t_{on} + t_{off}}{\frac{t_{on}}{t_{off}} + 1}$
t_{on}	$(t_{on} + t_{off}) - t_{off}$	$(t_{on} + t_{off}) - t_{off}$	$(t_{on} + t_{off}) - t_{off}$
C_T	$4.0 \times 10^{-5}\, t_{on}$	$4.0 \times 10^{-5}\, t_{on}$	$4.0 \times 10^{-5}\, t_{on}$
$I_{pk(switch)}$	$2I_{out(max)}\left(\frac{t_{on}}{t_{off}} + 1\right)$	$2I_{out(max)}$	$2I_{out(max)}\left(\frac{t_{on}}{t_{off}} + 1\right)$
R_{sc}	$0.3/I_{pk(switch)}$	$0.3/I_{pk(switch)}$	$0.3/I_{pk(switch)}$
$L_{(min)}$	$\left(\frac{V_{in(min)} - V_{sat}}{I_{pk(switch)}}\right) t_{on(max)}$	$\left(\frac{V_{in(min)} - V_{sat} - V_{out}}{I_{pk(switch)}}\right) t_{on(max)}$	$\left(\frac{V_{in(min)} - V_{sat}}{I_{pk(switch)}}\right) t_{on(max)}$
C_O	$9\frac{I_{out}t_{on}}{V_{ripple(pp)}}$	$\frac{I_{pk(switch)}(t_{on} + t_{off})}{8V_{ripple(pp)}}$	$9\frac{I_{out}t_{on}}{V_{ripple(pp)}}$

Figure 3.1.5. Power booster formula sheet[15]

The motor IC we used is L293D, a quadruple half-H driver, which is easy to implement. Each driver can drive two motors at the same time but it requires six GPIO pins in total. After we have the circuit design, we make the circuit as a double layer PCB, which is shown in Figure 3.1.6 and 3.1.7. We send the design to the company, PCBway, and ask them to manufacture the board for us.

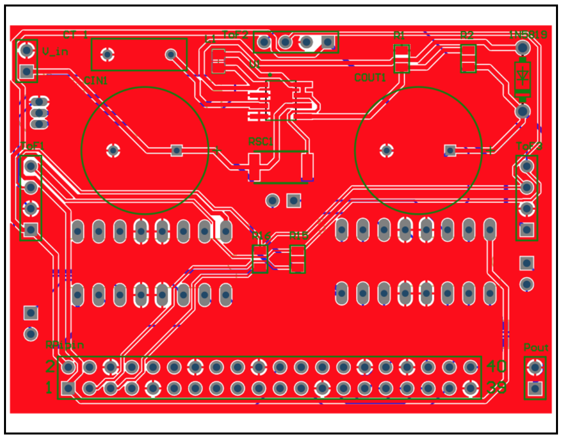


Figure 3.1.6. PCB top layer

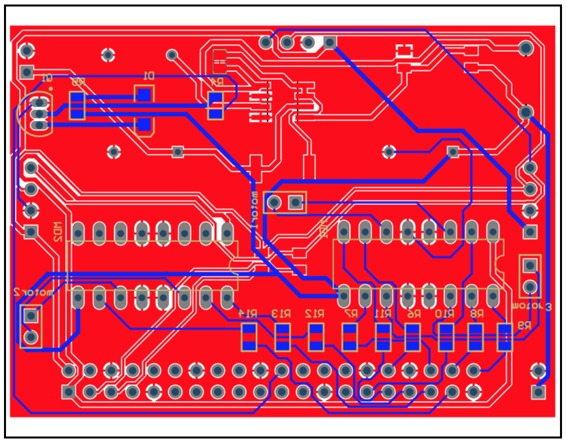


Figure 3.1.7. PCB bottom layer

After we received the board, we soldered the components to the board and tested its performance. (The list of the components was provided in Appendix D). Moreover, we used the same principle of the projector and built a new projector that can attach to the PCB directly, which is shown in Figure 3.1.8.

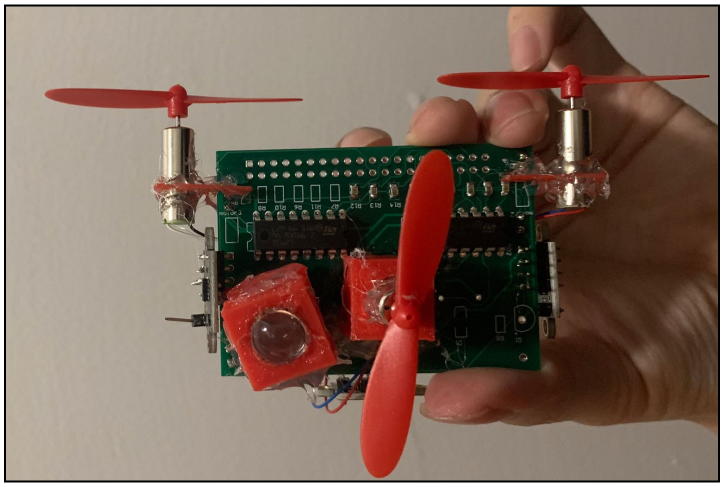


Figure 3.1.8. Complete system structure

3.2 Base Station Client Implementation

The base station client code is contained in base_station.py. It is responsible for three main tasks:

- maintaining the connection with the server running on the robot
- displaying information to the screen
- detecting inputs from the user in the form of mouse clicks and keyboard button presses

The base station was programmed using Python for the following reasons:

- It can easily interface with the server code running in Python on the robot
- Speed, which is typically the main drawback of Python, is not an issue for the relatively slow-moving blimp
- Python already contains many of the libraries needed to implement a simple client GUI communicating over WiFi

3.2.1 Client/Server Connection

The connection to the server on the client side is maintained using an object of the class `conn`. Initializing this object creates a socket object from the Python Socket library, which establishes a connection to the server. Currently, the server's IP address and port number for this connection must be hardcoded. This class also contains methods for exchanging data with the server. These methods format the request and accompanying data, if applicable, according to the server's expectations, send it, wait for the response, and decode the return message, if there is one, into a string that can then be parsed. Since these methods block while they are waiting to receive the return data, they are called in separate threads using Python's Threading library[16]. Since the server connection cannot be left open or it will time out, these threads must create a new `conn` object each time. The threads also take care of parsing the server's responses.

3.2.2 User Interface

As mentioned in previous sections, the client GUI is implemented using the Python PyGame library. This was chosen because it is a widely-available, easy-to-use graphics library for Python.

The code for laying out the display is all contained in the class `UI`. We considered creating some kind of screen object class for storing the data about each object on the screen, however it would require a significant rework of the code, and similar methods that we would like these objects to have, such as `draw`, would be so different for each object that having a single class would be almost meaningless.

Having a single class containing all of the drawn objects means that there are many constants in this class. Most of the objects on the screen have an x position, y position, width, and height (all in pixels). Generally, the x and y coordinates are stored for the upper-left corner of the object, but the centers of some are stored instead, depending on how they are drawn. Comments in the code specify this for each object. Many objects' positions and sizes are linked to those of nearby objects, so changing the size or position of one should not greatly impact the overall layout. Colors are also defined as constants for the class to allow for easy access. In accordance with PyGame requirements, these are (R,G,B) tuples.

Once every loop, the `update_screen` method is called. This method wipes the screen, then calls all of the draw methods to display the objects. This is slightly inefficient, since most things on the screen will stay constant from frame to frame. However, we aren't noticing it having trouble meeting its frame rate, so we did not feel the extra effort of calculating exactly which areas to erase and redraw was necessary. The frame rate is currently set to approximately 30 frames per second. This is likely faster than needed, since the blimp moves so slowly, so that number can be reduced if needed.

All of the draw methods in `UI` are preceded by two underscores, indicating they are helper functions and should not be called outside of `update_screen`. Most of the objects have a specific draw method for just that item. There are a couple, namely `__draw_table` and `__draw_switch`, that are used for multiple similar objects. They typically follow the same formula: draw the label text, the background, and any necessary text over the background using the predefined colors. Of particular note is `__draw_error_running`, which prints the red error/warning text at the bottom of the screen for three seconds. This is achieved using Python's built-in Time library[17].

3.2.3 Gathering User Inputs

The base station client is not simply an information display. It can also accept user input to send commands to the server or otherwise control the system. This input takes two forms: mouse clicks on the GUI and keyboard button presses. Fortunately, PyGame provides an interface for detecting both. These inputs trigger events in PyGame, and they can be accessed using `pygame.event.get()`, which returns a list of such events.

The most basic input is the mouse click. We use PyGame's built-in `MOUSEBUTTONDOWN` event to detect clicks. These events are triggered whenever the user presses the left mouse button, at which point the coordinates of the pointer are recorded. The interface contains various buttons and switches, so if one such event is detected within the bounds of one of these objects, we register a click on the object and process it accordingly. If the system is currently in waypoint mode, the interface also contains input text boxes for inputting the next waypoint. In this case, click events within these text boxes activate them. If the system is in manual mode, arrow buttons are activated by this event as well. In this mode, however, we also listen for `MOUSEBUTTONUP` events. These occur when the user releases the left mouse button, and we use them to stop the motors. These events in combination cause the motors to continue to run as long as the mouse is held down. Unlike some common button interfaces, however, the direction does not change if the user moves their mouse over a different directional arrow while still holding the mouse down. If this functionality is required, it will need to be implemented in a future version, perhaps by tracking the pointer's position in between `MOUSEBUTTONDOWN` and `MOUSEBUTTONUP` events while in manual mode.

While in waypoint mode, keyboard input is necessary as well for entering a new waypoint. For this, we use the `KEYDOWN` event, which records which key is pressed. In addition to the expected alphanumeric key inputs for the text boxes, we also implement backspace functionality to delete characters in the selected text box. In addition, the Enter key will trigger the Send button, just as though it had been clicked.

4 Tests and Results

4.1 Vicon Test

The first test was setting up the vicon tracking device and integrating it with the base station. The main issue we faced during this part is getting the IP address for the vicon lab. On further investigation we realised that the vicon lab was hooked to its own private network and our blimp was using Wi-Fi which couldn't be integrated with the vicon lab network.

4.2 Base Station Test

The first test of the base station that consists of the motor control was completely operational. In the second test, we integrated the Pi camera and OpenCV to get location estimates and display it on the

GUI. The station will receive the localization data once the server has detected the Apriltags. Also, the station is able to read the flight-sensor value and display on the user interface.

4.3 Motor Circuit Test

The motors are installed for maneuvering the blimp. They are currently controlled completely manually from the base station. The initial test proves that when the balloon is at the equilibrium position the motors work perfectly.

4.4 Projection System Test

The initial test proved that the arrow when projected from a lower altitude displays a clear sharp image on the floor. However when the altitude increases the image quality degrades. To remedy this we tweaked the design and added a more powerful LED that helped us gain a clearer image.

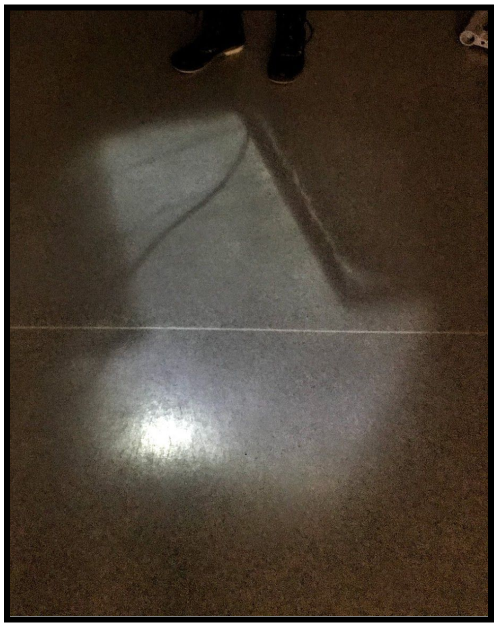
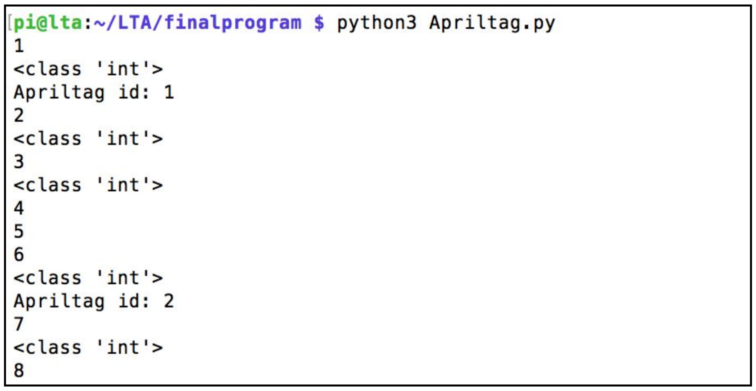


Fig 4.4 Projected arrow image

4.5 AprilTag Detection Test

We built an AprilTag detection program by using the OpenCV library developed by AprilRobotics. When we run the program on the Raspberry Pi Zero W, the Pi camera will start to capture the image every second. Then, the program will convert the image to grayscale and crop the square image which looks like a possible tag. Once the program confirms the tag number, it will return the tag ID value which shows in Figure 4.5.1. Then, the server will search the position that corresponds to the tag ID and send the location to the base station once the base station sends out the request data message.

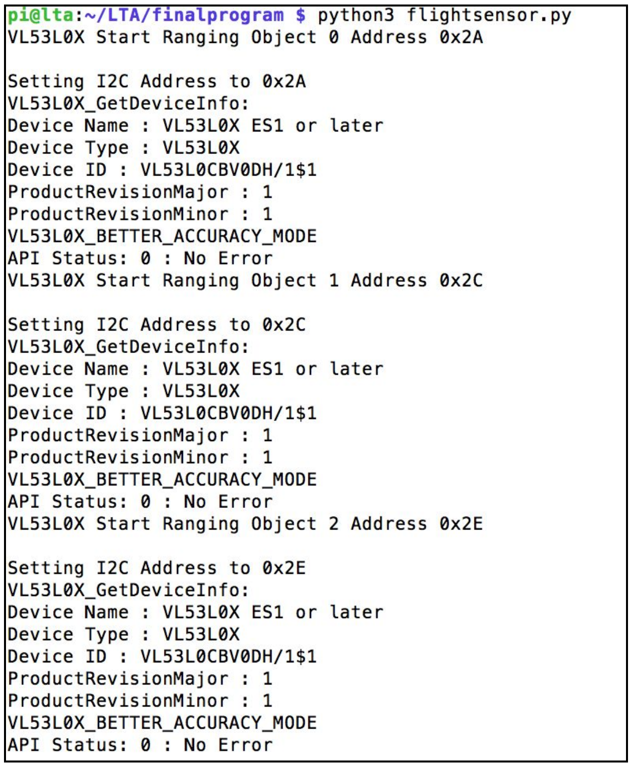


```
pi@lta:~/LTA/finalprogram $ python3 Apriltag.py
1
<class 'int'>
Apriltag id: 1
2
<class 'int'>
3
<class 'int'>
4
5
6
<class 'int'>
Apriltag id: 2
7
<class 'int'>
8
```

Figure 4.5 Output of the AprilTag detection program

4.6 Time-of-Flight Sensor Test

We used ST's VL53L0X time-of-flight distance sensor library that makes it simple to configure the sensor and read range data. However, three flight sensors are identical, so they have the same I2C address. Therefore, we need to set a GPIO pin for each sensor to act as a control in order to change their I2C address, which can be seen in Figure 4.6.1. After the program runs, each sensor will detect the distance between the obstacles and the sensor. The library reader will return the distance value in millimeters. The result shows in Figure 4.6.2.



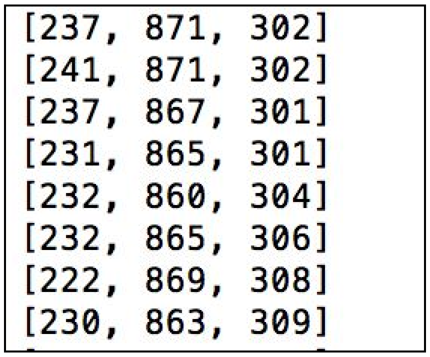
```
pi@lta:~/LTA/finalprogram $ python3 flightsensor.py
VL53L0X Start Ranging Object 0 Address 0x2A

Setting I2C Address to 0x2A
VL53L0X_GetDeviceInfo:
Device Name : VL53L0X ES1 or later
Device Type : VL53L0X
Device ID : VL53L0CBV0DH/1$1
ProductRevisionMajor : 1
ProductRevisionMinor : 1
VL53L0X_BETTER_ACCURACY_MODE
API Status: 0 : No Error
VL53L0X Start Ranging Object 1 Address 0x2C

Setting I2C Address to 0x2C
VL53L0X_GetDeviceInfo:
Device Name : VL53L0X ES1 or later
Device Type : VL53L0X
Device ID : VL53L0CBV0DH/1$1
ProductRevisionMajor : 1
ProductRevisionMinor : 1
VL53L0X_BETTER_ACCURACY_MODE
API Status: 0 : No Error
VL53L0X Start Ranging Object 2 Address 0x2E

Setting I2C Address to 0x2E
VL53L0X_GetDeviceInfo:
Device Name : VL53L0X ES1 or later
Device Type : VL53L0X
Device ID : VL53L0CBV0DH/1$1
ProductRevisionMajor : 1
ProductRevisionMinor : 1
VL53L0X_BETTER_ACCURACY_MODE
API Status: 0 : No Error
```

Figure 4.6.1. Initial I2C address setup of the TOF sensors

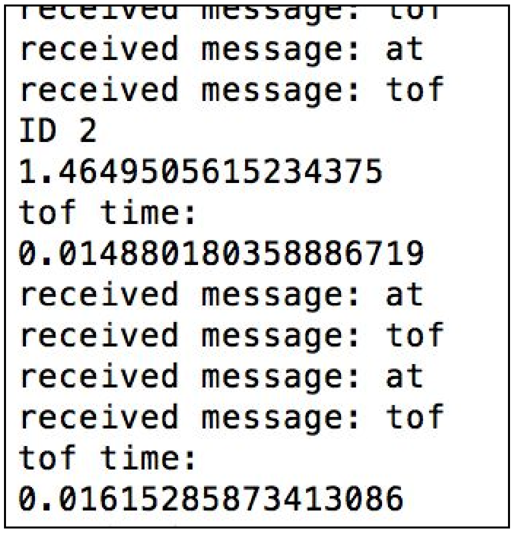


```
[237, 871, 302]
[241, 871, 302]
[237, 867, 301]
[231, 865, 301]
[232, 860, 304]
[232, 865, 306]
[222, 869, 308]
[230, 863, 309]
```

Figure 4.6.2. Output results from the TOF sensors [left, front, right]

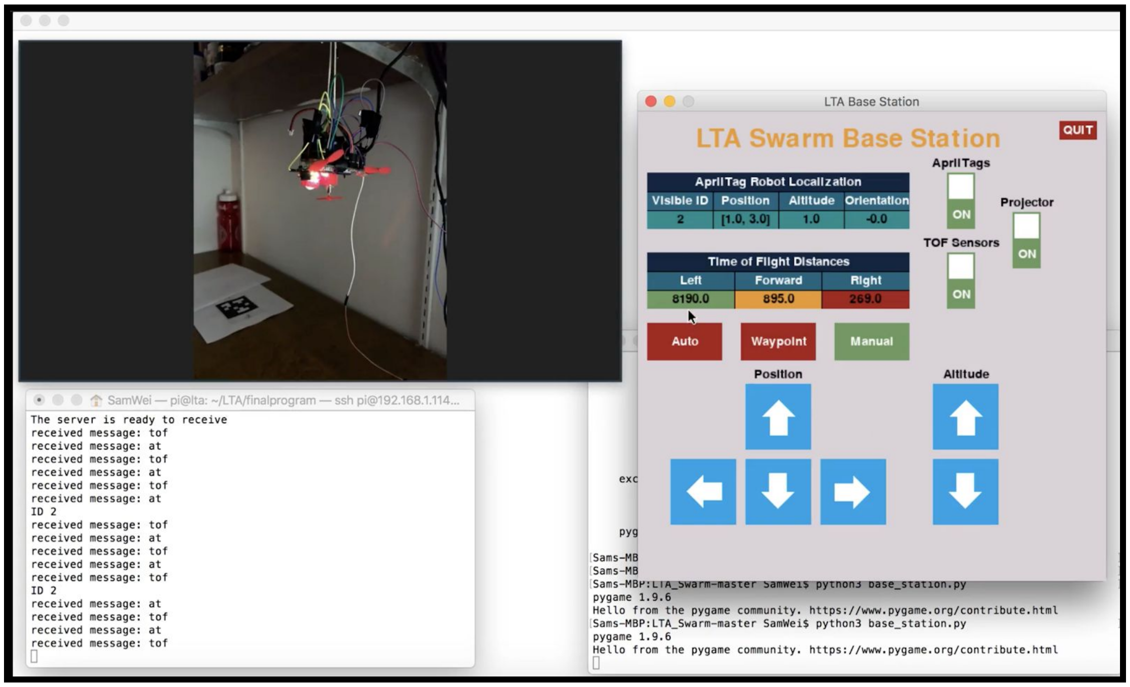
4.7 Complete Flight Test

After three main programs (motors, TOF sensors, and AprilTags) worked properly, we modified the server control program by adding these three programs as functions in the while loop. When the program operates, it will warm up each sensor and the camera. Once all the parts are set, it will show the message “The server is ready to receive”. Then, we can run the base station program in the other terminal to receive data and control the motors. However, we noticed that there was a time delay caused by the AprilTag detection function, whose elapsed time is about 1.5 seconds. Also, the elapsed time for the reader of flight sensors is 0.015 seconds, which might cause a small delay if the base station requests the data with high frequency. Therefore, we set the speed of requesting data to be once every second to prevent the program from having any segment faults.



```
received message: tof
received message: at
received message: tof
ID 2
1.464950561523437 5
tof time:
0.014880180358886719
received message: at
received message: tof
received message: at
received message: tof
tof time:
0.01615285873413086
```

Figure 4.7.1 The result output of the server control.



SamWei — pi@lta: ~/LTA/finalprogram — ssh pi@192.168.1.114...

```
The server is ready to receive
received message: tof
received message: at
received message: tof
received message: at
received message: tof
received message: at
ID 2
received message: tof
received message: at
received message: tof
received message: at
received message: tof
ID 2
received message: at
received message: tof
received message: at
received message: tof
```

AprilTag Robot Localization			
Visible ID	Position	Altitude	Orientation
2	[1.0, 3.0]	1.0	-0.0

Time of Flight Distances		
Left	Forward	Right
8190.0	895.0	269.0

```
exc

pyg
Sams-MB
Sams-MB
Sams-MBP:LTA_Swarm-master SamWei$ python3 base_station.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
Sams-MBP:LTA_Swarm-master SamWei$ python3 base_station.py
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
```

Figure 4.7.2 Screenshot from final video demonstration[1]

[1] Video located at https://drive.google.com/file/d/1BPuuGzSPC_zdGDF796BpymZ_HwlwO-LF/view?usp=sharing

5 Known Issues

5.1 AprilTag Detection

One issue we noticed after the complete integration of the system is that AprilTag detection currently takes a couple of seconds. However, there are ways in which we can improve this issue. As a part of the future work on the system, we can decrease the resolution of the Raspberry Pi camera as the tags being detected utilize large black and white chunks. We do not need to resolve anything more finely than the apparent size of those blocks. Currently, we are using the default camera resolution which is much higher than required for this purpose. Furthermore, to increase the speed of detection, we can use a smaller family of AprilTags. The one we are using right now belongs to the tag family 36h11, but for our model we can easily switch to the tag family 16h5. It has larger blocks which will provide better visibility in low resolution, thereby increasing the speed of detection and decreasing the required resolution of the camera.

5.2 Power Requirements

The main aim of the project is to build a lighter-than-air robotic drone. For the initial prototype we use a Lithium Ion polymer battery that weighed around 32 grams. This provided enough power to conduct all the experiments and support the load of running all the hardware, however it is too heavy to be included in the final design. Thus, we switched to a lighter, 10.5 gram Lithium Ion battery. The only issue with this was that it could not run the blimp for long durations. Therefore, we switched to using venom fly batteries which need a booster circuit to step up the voltage from 3.7V to 5V but can be used for longer durations.

5.3 Blimp Envelope

Initially, the design prototype that we were working with used a one foot balloon which could not lift our payload. We worked on crafting our own balloon envelope with mylar sheets to get a customized design, but that caused difficulties as the sealing process was taxing, and even a minor error would cause the balloon to deflate and the system to collapse. Instead, we finalized a multi-layer balloon “stack” design which enabled us to gain greater lift by stacking two 2-foot balloons.

5.4 Projection Image

The projection system displays a clear arrow at a lower altitude, but the image becomes blurry as the altitude increases. We have narrowed down the issue to the focal length of the lens that we are currently using. As a part of future improvements, we can look into using a different set of LEDs and larger focal length lenses which will help the image converge clearly even from higher altitudes.

5.5 Integration of PCB

We encountered a few issues with the PCB that could be improved in later versions. First, there's a problem in the power booster circuit that prevents the voltage from being boosted from 3.7 V to 5 V successfully. We believe that the reason may be that the components we selected did not match the requirements. It will have to be fixed in the next prototype. Second, there are some improvements that could be made to the TOF sensor interface. In our current design, there are resistors that go from the SCL and SDA pins on the sensors to the Raspberry Pi, but they are not necessary. Also, the pin connections from the sensors to the PCB were reversed, so we are currently using wires to connect them. These pins should be flipped to eliminate these unnecessary wires. Third, the dimension of the LED footprint did not match the real component, so it should be adjusted. Finally, the orientation of the Raspberry Pi 40-pin connector was reversed. It will need to be flipped to connect the Pi directly to the PCB without wires.

6. Future Work

Some future functionality has been implemented on the base station but is not yet completed on the robot itself. First, we currently only have the manual control mode fully functional. The base station has options for waypoint and fully-automatic control, but the robot is not yet capable of autonomous flight. Second, there are a number of enable switches on the base station interface. Currently, these only prevent the base station from requesting certain data. It would be beneficial for turning these switches off to also prevent that portion of code from running on the robot. This would reduce energy consumption and make the rest of the robot's code run faster. Third, the base station expects a distance and orientation when receiving AprilTag data. Currently, only dummy information is being sent for this. If this capability were implemented on the robot, the base station could display its altitude and orientation, allowing a better localization estimate. Finally, the base station allows for the boxes displaying the TOF sensor measurements to change color based on the proximity of an obstacle. It is expecting the robot to calculate this however, which it does not. Implementing this would allow the user to more easily see when the robot might be in danger of a collision.

Aside from these improvements, the robot could also be used for other applications in the future. It could be integrated into a swarm of drones that can be deployed in the emergency situation such that it can cover a greater range in less time. The onboard camera could also be used for applications other than AprilTag detection. One such application that was discussed but not implemented yet was to use it to provide choreography assistance to visually impared dancers. The camera can be used to detect the position of the dancers and identify if they are doing something wrong. This information can then be provided to the dancers using bluetooth headsets or speakers.

7. Conclusion

We have achieved our primary goal of making a prototype that is able to fly, localize itself, detect obstacles, and project an image. We were able to significantly reduce the mass of the payload, and with a few minor improvements to the PCB, this could be reduced even further. We believe this prototype will serve as a strong base for future work on the system.

Acknowledgement

We would like to thank Professor Joseph Skovira, Professor Kirsten Petersen and Nialah Wilson for their advice on our project. They provided us with useful insights that helped us make our project unique and increase the commercial value. Professor Skovira always responds to our question with careful guidance and great enthusiasm. Professor Kirsten Peterson is filled with creative solutions to any hardware difficulties. Nialah helped us with insightful suggestions that helped us better understand and further advance our project. In addition, we would like to thank Katie Bradford and Zoe Du for their advice and insights on PCB design and printing, and AprilTag implementation respectively. We truly appreciate all the help and encouragement we received from our mentors and peers for the project.

References

[1] Raspberry Pi Foundation. (n.d.). Raspberry Pi Zero W. Retrieved from https://www.raspberrypi.org/products/raspberry-pi-zero-w/

[2] Raspberry Pi Foundation. (n.d.-b). Camera Module V2. Retrieved from https://www.raspberrypi.org/products/camera-module-v2/+

[3] Adafruit Industries. (n.d.). PowerBoost 500 Basic - 5V USB Boost @ 500mA from 1.8V+. Retrieved from https://www.adafruit.com/product/1903

[4] ON Semiconductor. (2019). *MC34063A, MC33063A, SC34063A, SC33063A, NCV33063A Datasheet*. Retrieved from https://www.onsemi.com/pub/Collateral/MC34063A-D.PDF

[5] STMicroelectronics. (2018). *VL53L0X Datasheet*. Retrieved from https://www.st.com/resource/en/datasheet/vl53l0x.pdf

[6] Adafruit Industries. (n.d.-a). Adafruit DRV8833 DC/Stepper Motor Driver Breakout Board. Retrieved from https://www.adafruit.com/product/3297

[7] Texas Instruments. (2016). *L293x Quadruple Half-H Drivers Datasheet*. Retrieved from http://www.ti.com/lit/ds/symlink/l293.pdf

[8] duckietown. (2020, March 19). apriltags3-py. Retrieved from https://github.com/duckietown/apriltags3-py

[9] OpenCV. (2020). OpenCV. Retrieved from https://opencv.org/

[10] Multicomp. (2012). *1W High Power LED Datasheet*. Retrieved from http://www.farnell.com/datasheets/1636581.pdf

[11] Wholesale LED Lens. (n.d.). 10pcs x 10MM LED Optical Lens Smooth Convex Lens for 1W 3W LED Lens . Retrieved from https://www.amazon.com/10pcs-optical-lens-smooth-convex/dp/B00WFUEJJG

[12] Gibbs, K. (2020). *Projector* [Diagram]. Retrieved from http://www.schoolphysics.co.uk/age11-14/Light/text/Projector/index.html

[13] PyGame. (2020). PyGame. Retrieved from https://www.pygame.org/

[14] Python Software Foundation. (2020). Socket. Retrieved from https://docs.python.org/3/library/socket.html

[15] Gupta, S. (2018, July 17). *Calculating the Components' Values for Boost Converter* [Table]. Retrieved from https://circuitdigest.com/electronic-circuits/3.7v-to-5v-boost-converter-circuit-diagram

[16] Python Software Foundation. (2020b). Threading. Retrieved from https://docs.python.org/3/library/threading.html

[17] Python Software Foundation. (2020c). Time. Retrieved from https://docs.python.org/3/library/time.html

Appendix A - Code Structure

Our code is located at https://github.com/acn55/LTA_Swarm_Final_SP20

Files needed on base station PC:

- base_station.py: code to run the base station GUI
- up_arrow.png, down_arrow.png, left_arrow.png, right_arrow.png: image files used in the base station GUI

Files needed on Raspberry Pi:

- Apriltag.py: code for running the Apriltags machine learning detection from the OpenCV optimized library
- Camera.py: function fro initialize the camera setting
- Control.py: function for operating motors in different directions
- flightsensor.py: functions for operating and reading data from time-of-flight sensors
- LED.py: functions for operating the projector
- servercontrol.py: top level file for operating robot
- VL53L0X.py: library implementing the VL53L0X (TOF) platform-specific i2c functions through callbacks to the python SMBus interface
- apriltags folder: contains library code needed for interpreting AprilTags

Appendix B - Base Station & Robot Control Code Installation

Installing and running robot code

1. Download required files for the robot listed in Appendix A to the Raspberry Pi
2. Install the time-of-flight sensor code by running the following command:
   ```
   pip3 install git+https://github.com/naisy/VL53L0X_rasp_python.git
   ```
3. Install OpenCV by running the following command:
   ```
   sudo apt install python3-opencv
   ```
4. Install and set up the AprilTags code by running the following commands in order:
   ```
   pip3 install dt-apriltags
   git submodule init
   git submodule update
   make build PYTHON_VERSION=3
   ```
5. To start the code, navigate to the directory containing servercontrol.py and run the following command:
   ```
   python3 servercontrol.py
   ```

Installing and running base station code

1. Download required files for the base station listed in Appendix A to a PC
2. Install NumPy by running the following command:
   ```
   pip3 install NumPy
   ```
3. Install PyGame by running the following command:
   ```
   python3 -m pip install -U pygame --user
   ```
4. Ensure that the variables `server_ip` and `server_port` on lines 757 and 759 of base_station.py match the IP address of the Raspberry Pi and the port number being used, respectively
5. Make sure the server is already running on the Raspberry Pi
6. Run the base station code by running the following command from the directory containing base_station.py:
   ```
   python3 base_station.py
   ```

Appendix C - Weight Budget

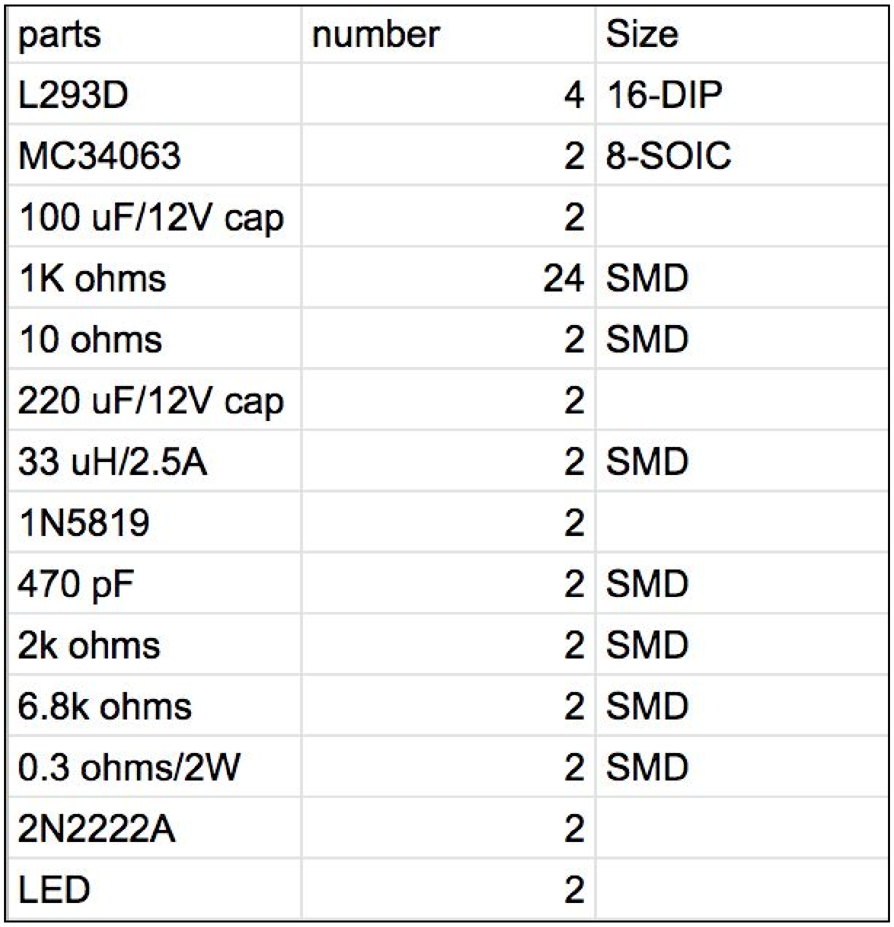
First prototype:

Components	**Mass Per Unit [g]**	**Quantity**	**Total Mass [g]**
1. Raspberry Pi Zero W [1]	9.8	1	9.8
2. Adafruit DRV8833	1.5	1	3
3. Lithium Ion Polymer Battery (500 MA)	10.2	1	10.2
4. Boost converter [3]	3	1	3
5. Motor with propeller	5.7	3	17.1
6. Projector	5.75	1	5.75
7. Blimp platform	7	1	7
8. Mini protoboard	2.3	1	2.3
9. Jumper Cables	1	24	24
10. Transistor n2222a	0.8	1	0.8
11. Micro USB cable	10.5	1	10.5
Total			**93.45**

Second prototype:

Components	**Mass Per Unit [g]**	**Quantity**	**Total Mass [g]**
1. Raspberry Pi Zero W [1]	9.8	1	9.8
2. Power booster	3.1	1	3.1
3. Lithium Ion Polymer Battery (500 MA)	10.2	1	10.2
4. Pi camera [2]	6	1	6
5. PCB + motors + flight-sensors	39.01	1	39.01
6. Jumper Cables	10	1	10
7. Micro USB cable	3	1	3
Total			**81.11**

Appendix D - PCB Components



parts	number	Size
L293D	4	16-DIP
MC34063	2	8-SOIC
100 uF/12V cap	2	
1K ohms	24	SMD
10 ohms	2	SMD
220 uF/12V cap	2	
33 uH/2.5A	2	SMD
1N5819	2	
470 pF	2	SMD
2k ohms	2	SMD
6.8k ohms	2	SMD
0.3 ohms/2W	2	SMD
2N2222A	2	
LED	2	