DroneHook
Final Report

University of Notre Dame

Matthew Peine, David Hurley, Adam Orr, Konrad Rozanski, John Bannan, Nathan Sanchez

EE-41440
Professor Mike Schafer
3 May 2022

1 Introduction

a. The Problem

Recently, there has been a growing interest in package delivery through the use of drones. Various companies have set out to develop a functioning method that can not only be easily accessible by the general public, but that can also be approved by the Federal Aviation Administration (FAA) . For example, companies like Wing, Flight Forward and Prime Air are leading efforts to find this fully functioning method with the help of billion dollar companies like Google, Amazon and UPS. However, these companies primarily focus on the package delivery side of things rather than package retrieval. Drones are typically viewed as the last link in the supply chain (e.g. Prime Air's vision of drones involves drones flying out of distribution centers and delivering packages to the customers doorstep. Although drones are generally seen as a method for delivering packages, drone package retrieval could be critical in certain scenarios. Examples that demonstrate time-critical retrieval scenarios include the retrieval of test samples from patients who are unable to make it to regional testing centers or the package retrieval of medication from one field hospital to another that has a patient who needs the medication.

DroneHook mitigates the aforementioned issues such that we can make use of a fixed-wing's superior efficiency and cruise speeds. We propose developing a system capable of retrieving a package "in-stride" and continuing onto its destination with little to no delay or change in altitude. Our system draws significant inspiration from the Fulton surface-to-air recovery system. We hypothesize that a fixed-wing with our system is capable of outperforming a variable takeoff and landing (VTOL) air frame. A package with a balloon attached to it will be deployed. Through the utilization of transmitted GPS coordinates, a fixed-wing will fly into visual sight of the balloon and then use its on-board computer vision system to snag the line and reel in the package. This would substantially set the stage for future research and work in this direction.

b. High Level Description

Firstly, we tackled the creation of a balloon mechanism with associated sensors capable of positioning the system and quantifying the volatility of the environment. This data was broadcasted by a transceiver module that utilized the SPI communication protocol in order to transmit the data to the out-of-sight fixed-wing. Secondly, developed an on-board system that receives the broadcast sensor data and utilizes it to generate an appropriate path to the balloon and attached package. We then determined the appropriate distance for the computer vision system, OpenCV, to take over. Both the on-board and on-balloon mechanisms have their own custom designed boards in order to meet this specification.

The on-balloon system included a custom board with the GPS, barometer and transceiver modules in order to take in location information and send it to the on-board system. The GPS sends information at a baud rate of 9600 to the PIC32 then the transceiver, which then sends the information to the on-board system. The barometer sends the information in the same way. The barometer gives much more accurate altitude readings, and as this is incredibly important for our purposes, we deemed the device necessary.

The on-board system included our custom board with a barometer, transceiver, and one UART interface with a Raspberry Pi, with python code that parses the GPS data from the transceiver, and utilizes OpenCV for location correction. It also includes a PiCam for use in

OpenCV. This Raspberry Pi also uses a UART connection with a PixHawk Cube 4, which controls the plane. All of these devices operate at 9600 baud. The way this works is the transceiver receives data from the on-balloon system. Then the Pi receives this data along with the current barometer data and sends it to the PixHawk. MavLink on the PixHawk then takes that data and creates a waypoint. When the PixHawk reaches about 30 meters outside of the waypoint, OpenCV takes over and uses the balloon's relative positioning within the frame in order to set velocity waypoints to hit the balloon and pop it.

c. *Meet Expectations?*

In general, the on-balloon system was able to reach expectations. In order to save time, we did not end up using the barometer data, as the GPS does give altitude coordinates, albeit not quite as accurate. The system was able to send the data to the transceiver on the on-board system, and was able to be held up by the balloon. However, the balloon had to be quite large in order to hold the weight of the custom board. The only true problem we encountered is that one of our balloons flew away when the twine became untied on the airfield.

The on-board system was where we truly encountered problems. All of our subsystems worked separately. We ran a MavLink simulator that we were able to upload waypoints to, we were able to receive the data from the on-balloon GPS, and we were able to get the computer vision to work. The computer vision was able to output direction vectors in a very timely manner as well. However, we struggled to get the actual frame in the air despite all of this. We 3-D printed landing and takeoff gear in order to protect it. The landing gear was time-consuming to make, and was not super stable due to the custom print. Furthermore, when the plane actually launched, it first launched crooked and uneven, and then when it finally made it into the air, the wind ended up pushing it over, breaking the entire frame. While this did not necessarily have anything to do with the actual embedded systems we designed, it proved difficult for us to test each of the subsystems together for this reason. While our project was not able to fully work, we were able to get all of the subsystems to work separately and some together. In short, our subsystems worked exactly how we wanted them to, but the physical design of the plane prevented us from full integration of our design.

2 Detailed System Requirements

OR1.0: The system shall be able to fly the UAV from beyond visual line of sight (BVLOS) to within 10 feet of the balloon utilizing transmitted data and computer vision system to construct trajectory

OR2.0: The system shall consist of an on-balloon embedded system and UAV system which will interact via a computer vision system onboard the plane and an RF link

OR3.0: The one-way wireless link shall be capable of transmitting sensor data from the on-balloon system to the UAV when the two are separated by a distance of at least 2000 m

OR4.0: The UAV embedded system must fit within a 5 inch x 5in x 5in volume due to onboard space restrictions

OR5.0: The on-balloon mechanism must weigh less than 250 grams to satisfy the payload limit of 36 inch helium weather balloons

OR6.0: The computation executed by the on-balloon and UAV systems' microcontrollers shall be restricted to serial (i.e. I2C, UART, and SPI) communication with sensors and data aggregation

OR7.0: The on-balloon mechanism shall possess a footprint smaller than 3 inch by 3 inch so that it can be fully encased in a lightweight TPU case

OR8.0: The system shall be capable of operating in environments with winds speeds of 5 miles per hour (i.e. breezy conditions)

OR9.0: The UAV embedded system shall consume no more than 5% of the energy stored in the 5S 2P 15,000mAh bank of lithium polymer (LiPo) batteries used to power the systems onboard the fixed-wing

OR10.0: The computer vision system shall be capable of resolving the balloon against shades of backgrounds ranging from dark blue to light gray

OR11.0: The on-balloon and UAV embedded systems shall be capable of operating at a temperature range of 0 F to 110 F with optimized performance at moderate temperatures (i.e. 50 F)

OR12.0: The RF-transmitted and computer vision generated balloon data shall be updated at a rate of at least 40Hz to maintain safe generation of control sequences seeing that the cruise speed of the aircraft is roughly 16 ft/s

OR13.0: The UAV embedded system shall weigh below 90 grams such that its installation does not interfere with the balancing of the airframe

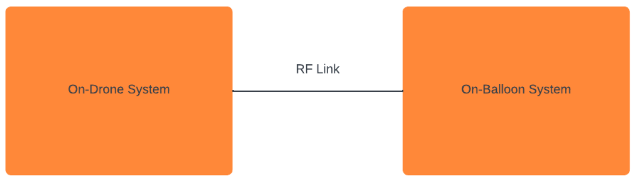
3 Detailed project description

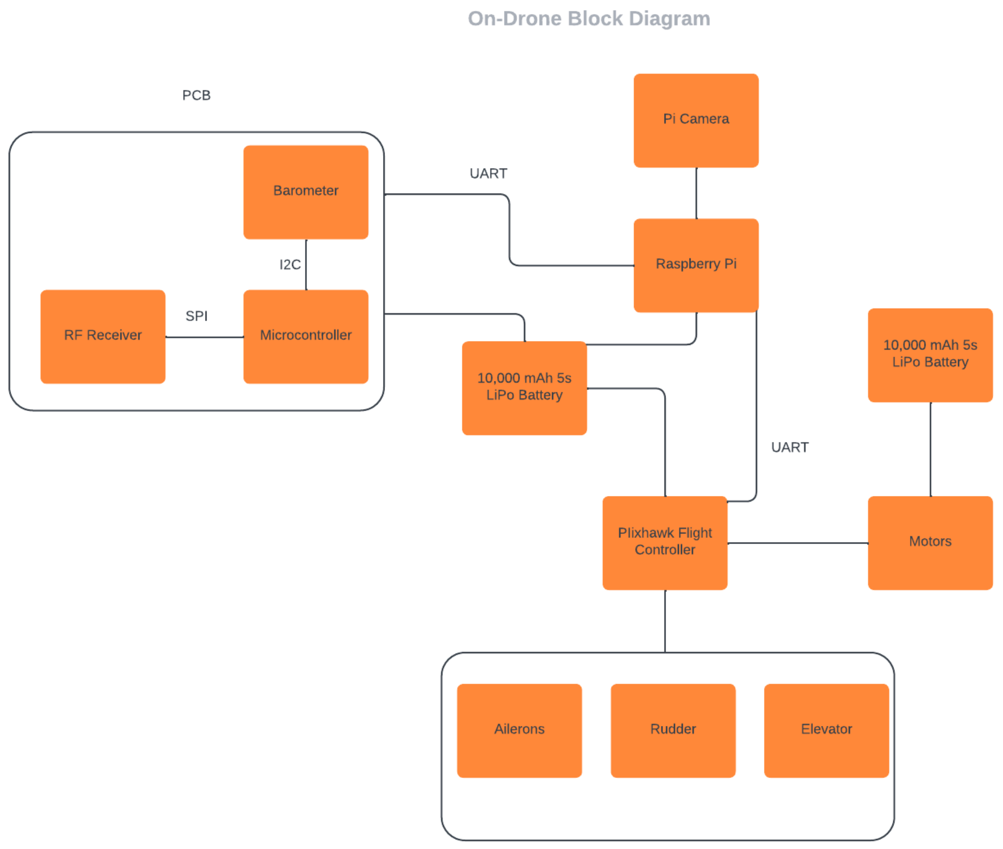
3.1 System theory of operation

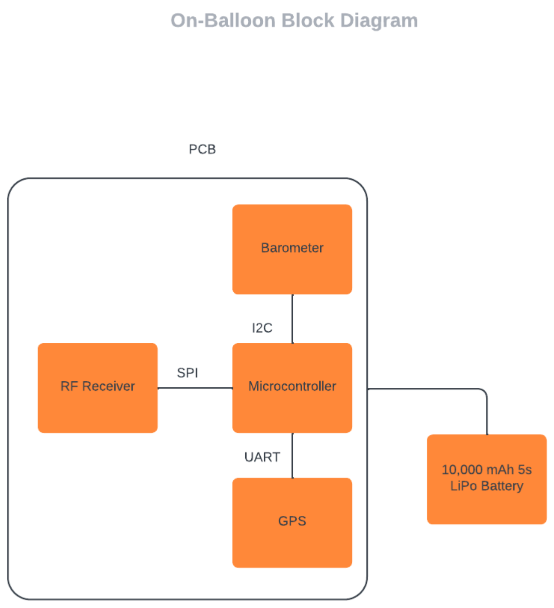
This system consists of two major components, one on a fixed wing drone and the other on a weather balloon. The overall concept is that the drone will locate the weather balloon and fly towards it until it completes a pass above it, and at that point it will return to the takeoff point and land. The communication between the balloon system and the drone system was through RF transceivers utilizing the LoRa protocol. The data being sent from the balloon was altitude, latitude and longitude. The on-drone flight controller uses that data to set a waypoint and controls the flight systems, including the motors, airelons, rudder and elevator, to direct the drone to that point. Upon being within ~200 feet of the set waypoint the system switched to computer vision. The pi camera was set to detect red spheres and located the center of the weather balloon. The pi mapped the x-axis of the camera to North and East and the y-axis to down to generate the appropriate velocity vector within a loop. That velocity vector was sent to the flight controller which implemented the appropriate control sequence to continually correct the flight path until the balloon is no longer detected, at which point the landing sequence was implemented.

3.2 System Block diagram

System Block Diagram




On-Drone Block Diagram
PCB
Barometer
I2C
RF Receiver
SPI
Microcontroller
UART
Pi Camera
Raspberry Pi
10,000 mAh 5s LiPo Battery
10,000 mAh 5s LiPo Battery
UART
Plixhawk Flight Controller
Motors
Ailerons
Rudder
Elevator



3.3 Detailed design of On Balloon Subsystem

On Balloon Subsystem Requirements

- **SR1.0:** The on-balloon embedded system shall transmit relevant sensor data as described in **SR2.1** to the UAV so that the balloon can be located and flown to
  - **SR1.1:** The system shall be powered by a Li-Ion battery
    - **SR1.1.1:** The battery shall be capable of powering the data transmitter system referenced in **R1.2**

        - **SR1.1.2:** The system shall stay full functional for a minimum of 10 minutes on a single fully charged battery
        - **SR1.1.3:** The battery shall not exceed 50 grams
    - **SR1.2**: The data transmitter system shall include:
        - **SR1.2.1:** An RF transceiver for long range, rapid transmission of on-balloon data
        - **SR1.2.4:** A GPS module for gathering two-dimensional coordinate data (i.e. x and y axes) with a resolution of at least 20 feet
    - **SR1.3:** The balloon itself shall be a distinct and unnatural color which can be easily recognized by the computer vision system against the background of the various shades of the clouds and sky
    - **SR1.4:** The system shall possess a microcontroller that fuses the data from the various on-balloon sensors and transmits via an RF transceiver
        - **SR1.4.1:** The microcontroller shall possess serial interfaces to accommodate the devices specified in **R1.2**
        - **SR1.4.2:** The microcontroller shall be low power and robust to operate in various weather conditions as specified in the overall requirements **OR10.0**, **OR11.0**, and **OR14.0**

On Board Drone Subsystem Requirements

- **SR2.0:** The on-board embedded system shall receive relevant sensor data from the balloon and on-board computer vision
    - **SR2.1:** The relevant received data shall be the x and y coordinates from the GPS module, the altitude or z-coordinate from the barometer.

        - **SR2.1.1:** An on-board Raspberry Pi will be responsible for the onboard handling and computation of the above data.
        - **SR2.1.2:** The Raspberry Pi shall utilize this data to compute an autonomous flight pattern and control sequence.
    - **SR2.2:** Autonomous flight control shall be responsible for implementing this autonomous flight pattern and control sequence and controlling the motors and other flight control features to realize the approach towards the balloon.

3.4 Operation of the Subsystems

Power System

The power system for our balloon board needed to supply 3.3V DC to all of the devices on the board. For this, we chose the LD1117 3.3V linear regulator. The input for this device was a 7.2V LiPo battery. We had to increase the size of the battery because we found out the voltage dropout of the LD1117 was too high (1.2V) to support the original 3.7V single cell battery. However, once this change was made, the device was able to supply a constant 3.3V. Figure A shows the pinouts for the LD1117, while Figure B shows the device's layout on the board.

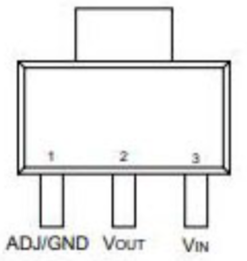


Figure A. LD1117 Pinout

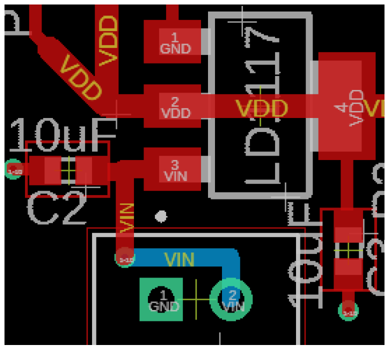


Figure B. LD1117 PCB layout

10uF capacitors were used on the input and output of the device, as stated as the recommended layout in the data sheet. A multimeter was used to ensure the device supplied 3.3V everywhere on the board. An LED was also used to indicate power was on.

The plane board also had a larger 5V linear regulator daisy chained to supply both the devices on the board as well as the Raspberry Pi via a USBC cable. This was powered by a 5 cell LiPo battery. The specific regulator was the LM1084. Figure C shows the typical layout of the device for providing 5V, which was used to find the resistor values.

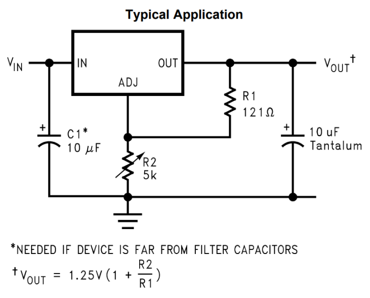

Typical Application
V_{IN}
IN
OUT
ADJ
$V_{OUT}^{\dagger}$
R1 121Ω
C1* 10 μF
R2 5k
10 uF Tantalum
*NEEDED IF DEVICE IS FAR FROM FILTER CAPACITORS
$^{\dagger}V_{OUT} = 1.25V(1 + \frac{R2}{R1})$

Figure C. LM1084 recommended layout

Figure D shows the PBC layout on the board.

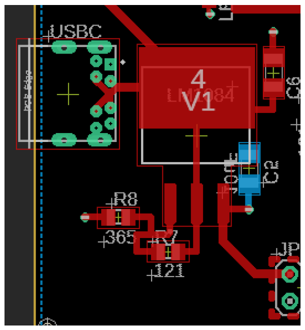


Figure D. LM1084 PCB layout

PIC32 Microcontroller

For the balloon and drone on board system we decided to choose the PIC32MX270F256B microcontroller. We decided to go with this model for several reasons. The first reason was due to the group's overall familiarity with the PIC32 family of microcontrollers and using them with embedded systems and external peripherals as well as using the UART and SPI protocols with these family of devices. Furthermore, the microcontroller we chose had UART and SPI capabilities that we required in order to communicate information using our RF transceivers, on board GPS systems, and the Raspberry PIs. We also had originally planned to use a barometer in our design with the microcontroller using UART and I2C, however this idea was not implemented and instead we opted to just use the altitude data via the GPS. We chose the PIC32MX270F256B model in particular because it had fewest pins that we needed and so the other models of microcontroller had excessive amounts of pins that were not going to be used or required by our design specifications.

For software, the PIC32MX270F256B was programmed using MPLAB and a PIC32 programmer. The PIC32 software can be found on our website for the on board and balloon boards. On the balloon system the PIC32 took in UART serial data from the GPS and set it via the RF transceiver using SPI to the board on the drone. This data is sent to the RF receiver via LoRa. The PIC32 on the drone would then take the GPS information via the RF receiver through SPI and send it to the Raspberry PI for processing and parsing.

The subsystems for the microcontroller were all tested together, thus ensuring proper data was being sent and received from the microcontroller. We used the Saliae logic analyzer to determine that the SPI and UART pins were working. Below in Figure 1. is the output of the logic analyzer showing the UART and SPI functionality sending GPS data from the balloon system. More information involving each part in the subsystem can be found below in their respective sections.

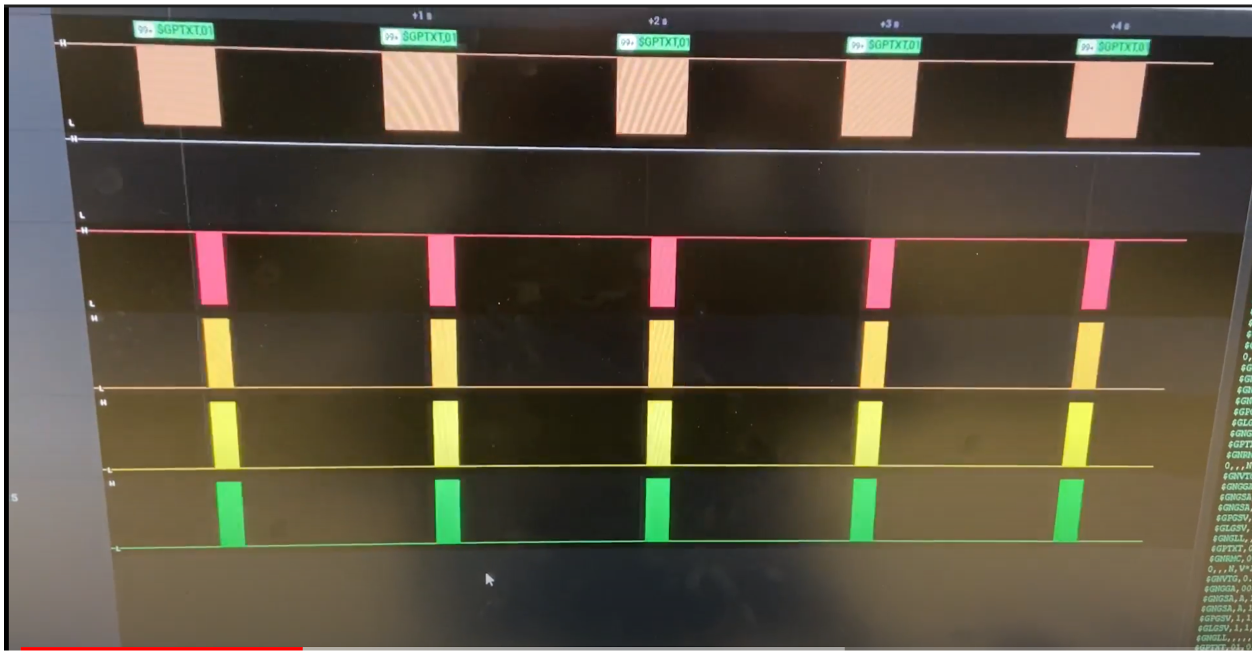


Figure 1. Logic Analyzer Output on UART and SPI communications for Balloon subsystem sending GPS Data

SX1276

LoRa was recommended for this project by Dr. Fay, and is a SemTech patented low-power wide-area communication technique. This device was used both on the balloon system and plane system. This was an ideal solution because of its frequency range (800 MHz – 1000MHz), long range (10km), and relative low amount of bandwidth. The device interfaced with the PIC32 via SPI. Below are the connections used for the subsystem.

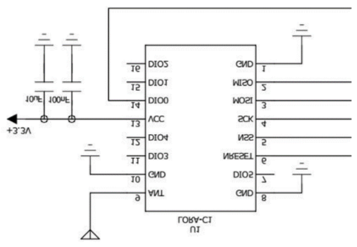


Figure 2. SX126 pinout connections

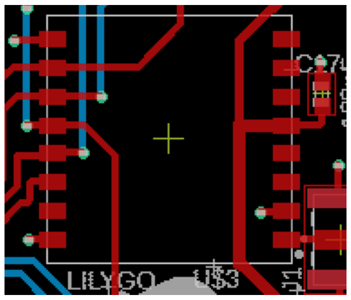


Figure 3. PCB SX1276 layout

The integration code between the PIC32 and SX1276 was written in C in the MPLAB environment. The essential connections to the PIC32 included SS (Pin 13), SCK (Pin 12), MISO (Pin 10), MOSI (Pin 11), and RESET (Pin 14). The Antenna pin connected to an SDA connector for the antenna. Figure 9 shows the transmission sequence that the code was structured around, outlined in the datasheet for the SX1276.

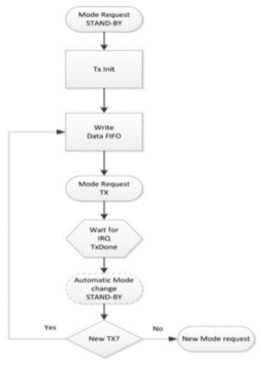


Figure 9. LoRa® Modulation Transmission Sequence.

Initially, the subsystem was tested by sending a packet of data to a receiver connected to an Arduino. The logic analyzer was also used to compare the output waveforms to the waveforms when the transceiver was initially used with an Arduino.

LC86L GPS

The LC86L was the choice for the GPS module for the balloon system. This was a new device that had to be ordered because our original choice of the PA1010D was out of stock. However, the LC86L operated very similarly to the PA1010D, so the only thing that had to change was the pinout layout on the schematic diagram. Figure 12 shows the pin diagram for the LC86L. The device communicated to the PIC32 via UART and was coded in C in the MPLAB environment.

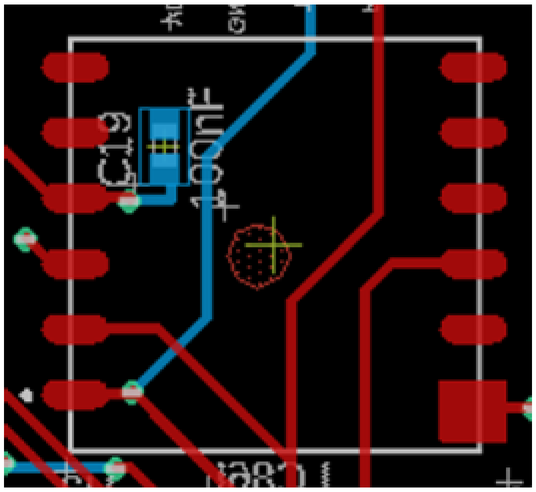


Figure 11. GPS PCB layout

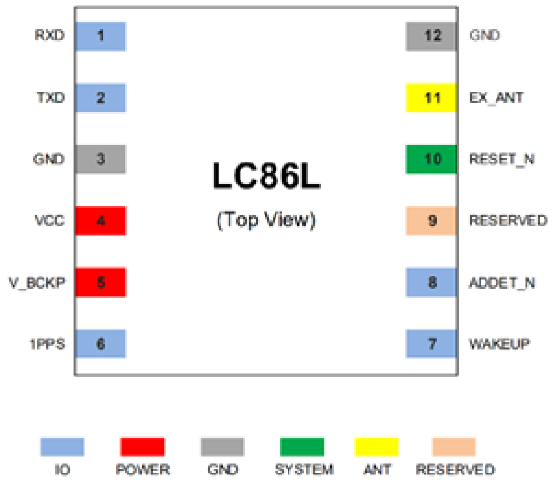


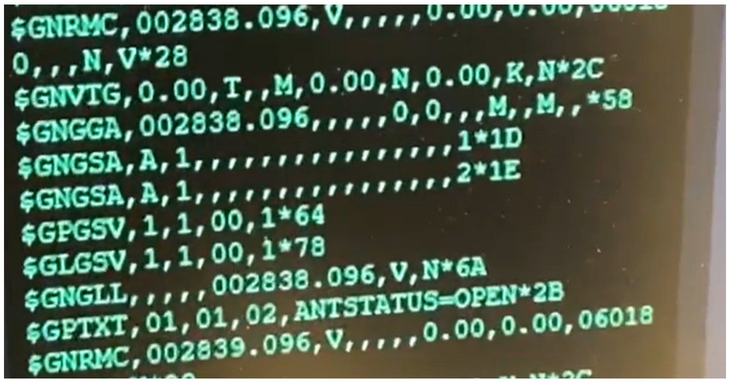
Figure 12. LC86L Pinout

```
$GPGGA,181908.00,3404.7041778,N,07044.3966270,
W,4,13,1.00,495.144,M,29.200,M,0.10,0000*40
```

Figure 13. NMEA sequence example

Once the device was powered on, it constantly sent over NMEA data. Figure 13 shows the output data sent from the LC86L. On the PIC32, we parsed the data to only obtain the $GNGGA sequence of the NMEA data. This includes the latitude, longitude, and altitude data of the balloon. This data would then be sent over to the plane via the SX2176 transceiver.

The GPS subsystem was tested prior to attaching the board onto the balloon. After powering on the board and connecting the board to the Logic Analyzer, a signal was detected from the board. The GPS outputs a signal with NMEA format GPS data.



```
$GNRMC,002838.096,V,,,,,0.00,0.00,
0,,,N,V*28
$GNVTG,0.00,T,,M,0.00,N,0.00,K,N*2C
$GNGGA,002838.096,,,,,0,0,,,M,,M,,*58
$GNGSA,A,1,,,,,,,,,,,,,,,,1*1D
$GNGSA,A,1,,,,,,,,,,,,,,,,2*1E
$GPGSV,1,1,00,1*64
$GLGSV,1,1,00,1*78
$GNGLL,,,,,002838.096,V,N*6A
$GPTXT,01,01,02,ANTSTATUS=OPEN*2B
$GNRMC,002839.096,V,,,,,0.00,0.00,06018
```

Figure X

Shown above in Figure X is the output data from the GPS. Although it had various NMEA GPS outputs, the only one of interest was the line followed by the '$GNGGA'. The GNGGA line outputs various data points; however, the only data used was latitude, longitude and altitude. With the output data that is shown above, the GPS subsystem was successfully able to output the data that was needed.

Raspberry Pi4

In order to interface on-board with the custom board, the PiCam, and the PixHawk Cube4, we utilized a raspberry pi with Raspian flashed on it. While the interface with the camera was built-in and rather straightforward to configure, the other two interfaces were a little more complicated. The connection to the board and the PixHawk required two UART connections. In order to accomplish this, we utilized the GPIO pins on the raspberry pi, and enabled UART0 and UART3. UART0 utilizes GPIO pins 14 and 15, and UART3 utilizes GPIO pins 4 and 7. We used UART0 to interface with the board at a baud rate of 9600 and UART3 to interface with the PixHawk at a baud rate of 115200. The PixHawk has multiple UART ports, and we used the second one.

The Pi also has the OpenCV and dronekit libraries installed onto it, as well as a script that uses both libraries in order to track the balloon and pop it. This Python script was responsible for intaking the information from the other applicable subsystems and acting upon that information in an appropriate manner. A copy of the script can be found in the appendix.

Vision System:

The vision system was tested in the lab prior to flying the drone. Using the camera that was attached to the front of the drone, the vision system was tested using the python script that followed the balloon and indicated to the drone where it should fly to. To test this, the camera was set up at the end of the hall with the python program running. Then, a red balloon was moved slowly towards the other end of the hall. The vision system was able to track the middle of the balloon and follow it around. Based on the x and y coordinates of the balloon center within the frame the vx, vy, and vz velocities were calculated. Those directions translate to North, East, and Down so the vector had to be rotated depending on the plane's heading as shown in the code snippet below. Based upon the coordinate plane, the demo also outputted left, right, up and down, in order to make the demo easier to understand for a general audience.

```
 vx = ((7-7*(x/xmax)) * (cos(theta) - sin(theta)))
  vy = 7*((x/xmax) * (sin(theta) + cos(theta)))
  vz = (y/ymax) * -3.5
```

Drone Build

In addition to the embedded systems on board the balloon and plane, there were also physical requirements to the subsystem. These included the frame and control system of the drone as well as the CAD printed parts for the drone itself. The drone itself was constructed of a styrofoam frame that was purchased online. The additional components we added to the frame included:

- Servos for thrust
- Propellers to attach to said servos
- Servos for operation of ailerons, rudder and elevator
- Pushrods to control sequences
- 3-D printed components:
    - Back landing skid made from TPU
    - Front landing gear (TPU) with wheels and a TPU brace
    - Socket for gear connecting to frame from PLA to allow for a margin of error.

4 System Integration Testing

4.1 Subsystem Testing

While we were not able to get a full system test due to the crash landing of the plane during test flight, we were able to test the subsystems working together. These systems included the computer vision, the MavLink flight simulator, and the GPS transceiver. Essentially, what we had to do to test it at a high level was run the MavLink simulator, wait for the board to send GPS coordinates, use the dronekit code on the raspberry pi to set a waypoint on the simulator once the coordinates are received and parsed on the onboard system, and once the plane is within about 50 meters, then switch to openCV and output what direction the plane needs to be corrected to.

4.2 Meeting Design Requirements

OR1.0: In the simulation the drone successfully reached the waypoint set by the GPS data and then successfully received the velocity vector from the computer vision system to redirect the plane towards the balloon.

OR2.0: This was implemented for the system.

OR3.0: GPS coordinates were received from a building or two away but did not test 2000 meters.

OR4.0: The system fit comfortably within the UAV.

OR5.0: The on-ballon mechanism was approximately 100 grams.

OR6.0: Transceiver usd SPI, barometer used I2C, and the rest used UART.

OR7.0: The system on the balloon was very small and held only the circuit board which was within the specification. Instead of a TPU case we opted for an open PLA tray to hold the PCB.

OR8.0: The wind was never below 8 mph during the weeks we were testing.

OR9.0: The UAV embedded system shall consume no more than 5% of the energy stored in the 5S 2P 15,000mAh bank of lithium polymer (LiPo) batteries used to power the systems onboard the fixed-wing

OR10.0: The computer vision system successfully detected the red balloon against any neutral background.

OR11.0: The system operated and performed between 40 F and 80 F during testing.

OR12.0: Our computer vision system achieved ~30 frames per second. This was the limiting factor and not the RF transmission.

OR13.0: The embedded system was below this threshold, and the plane did successfully takeoff. The frame and motors were rated for up to 25 total pounds our completed plane weighed ~16.7 pounds.

5 Users Manual/Installation manual

5.1 Installation

Hardware prerequisites:

- Flight controller with available telemetry or UART port
    - **Recommended:** Pixhawk Cube Orange
- 5V USB-C power supply
    - **Recommended:** 3A max current rating
- Airframe with capacity for Pi HQ Camera & 16mm lens
    - **Recommended:** Fixed-wing with ailerons, elevator, and rudder

Our system consists of three main installation components onboard the UAV: **(1)** Pi HQ Camera and 16mm lens, **(2)** Raspberry Pi 4 8GB, and **(3)** On-Drone Board. Beginning with the installation of **(1)**, the camera and lens should be mounted in the nose of the vehicle such **(A)** the view of the front is unobstructed by any part of the vehicle and **(B)** the center of the camera's view corresponds to the center of the vehicle from the front view. **(2)** can be installed anywhere within the vehicle's fuselage under the condition that the CSI cable from **(1)** can reach the connector on **(2)** and the USB-C power cable from **(3)** can reach the USB-C female connector on **(2)**. **(3)** can be installed anyplace within the airframe assuming that the UART cable from **(2)** to **(3)** can reach as well as the UART cable from **(2)** to the flight controller (presumed to be a

Pixhawk Cube Orange). The ground pin on **(3)** must be connected to any ground **(2)**. Likewise the TX pin on **(3)** must be connected to the RX on **(2)**.

5.2 Setup

Software prerequisites:

- Python 3 must be installed on the Raspberry Pi
    - **Recommended:** Python 3.9
- A laptop must have a ground control station installed
    - **Recommended:** Mission Planner
- The flight controller must have a ArduPilot installed
    - **Recommended:** ArduPlane 4.1.7

The first steps of setup require that all of the proper software be installed onto the On-Drone board, Pixhawk Cube Orange, and Raspberry Pi. Beginning with the On-Drone board, a PICKit3 programmer is used to flash **UAV.X** project onto the board. Next, the **PythonCode.txt** script found in the UAVIntegratedSystems directory must be installed on the Raspberry Pi. The packages listed as being required in the UAVIntegratedSystems.txt file must be installed before this script can be run. Finally, the OnBalloon.X project must be flashed on the board that will be hanging from the balloon. When everything has been successfully installed, the user should be able to run ArduPlane's native simulator and see the plane taking off. This simulator can be run be via the following two commands:

1. Python3 PROGRAM_NAME.py
2. sim_vehicle.py -v ArduPlane --console --map

5.3 Functionality

In order to make sure that the entire system works, there are a few fundamental things that must be tested. First of all, the plane must fly. Second, the GPS must send coordinates over the transceiver to the onboard system, the simulator must be able to send those coordinates and set a waypoint, and the computer vision system must output velocity vectors (or left right, up down) based upon the balloon's position within the frame.

In order to make sure the plane flies, upload a singular waypoint (does not matter what as long as it is close by) to the PixHawk and then make it land. If this works, the plane should be able to fly a mission.

In order to make sure the GPS works, take the balloon a decent distance from the on-board system and attempt to send the data over a transceiver. If it is received in one piece it works.

In order to test the simulator, run the commands and make sure waypoints are received and that it can complete a roundtrip. Run a multitude of other commands as well to make sure the connection works.

In order to test the computer vision system, take a balloon and make sure it prints the correct directions based on its relative position within the frame.

5.4 Troubleshooting

- On-Balloon GPS

The on-balloon GPS takes some time to locate the satellites and receive quality data. If the the system is not outputting the correct location, or no location at all, take the system to a place with good reception and minimal interference from trees and buildings. The GPS should start transmitting data within 10 minutes.

- On -Drone GPS

Follow the same procedure as outlined above.

- ESC Calibration

Prior to launch the user should ensure the motors used for thrust start at the same time. This can be accomplished by very slowly increasing the thrust on the remote control. If the ESCs (Electronic Speed Controller) are not correctly calibrated one motor will start rotating before the other. This can be resolved by following the steps outlined here: https://ardupilot.org/copter/docs/esc-calibration.html.

- Control Surface Calibration

Prior to launch the user should make sure that all of the control surfaces (aierelons, rudder, elevator) are centered. If this is not the case use the remote control to trim the appropriate control surface.

6 To-Market Design Changes

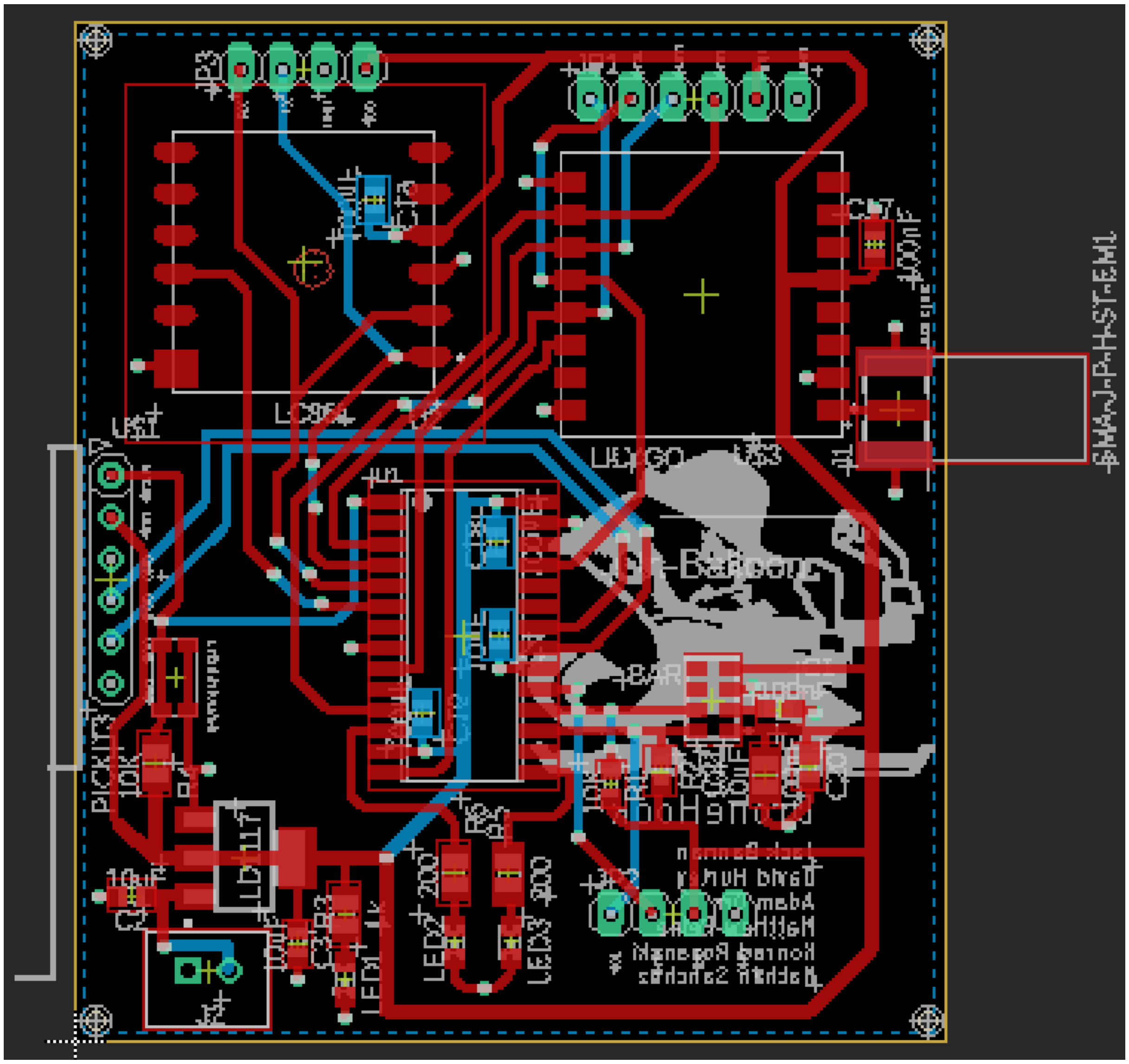
This project was very close to what our goal would be for a MVP (minimum viable product) for taking it to-market. The entire model would need to be scaled up to introduce the possibility of longer range solutions. Fiberglass and carbon fiber would substitute the styrofoam frame to increase the robustness of the system. Additionally we would design and implement the snatching mechanism. The purpose of this module would be to sntach the line connecting to the balloon and secure a package which is also connected to the line. This would allow for package retrieval and delivery without needing to land and thereby saving both time and energy. To further increase the range and efficiency of the drone we would eventually transition from a fully electric to a parallel hybrid system.

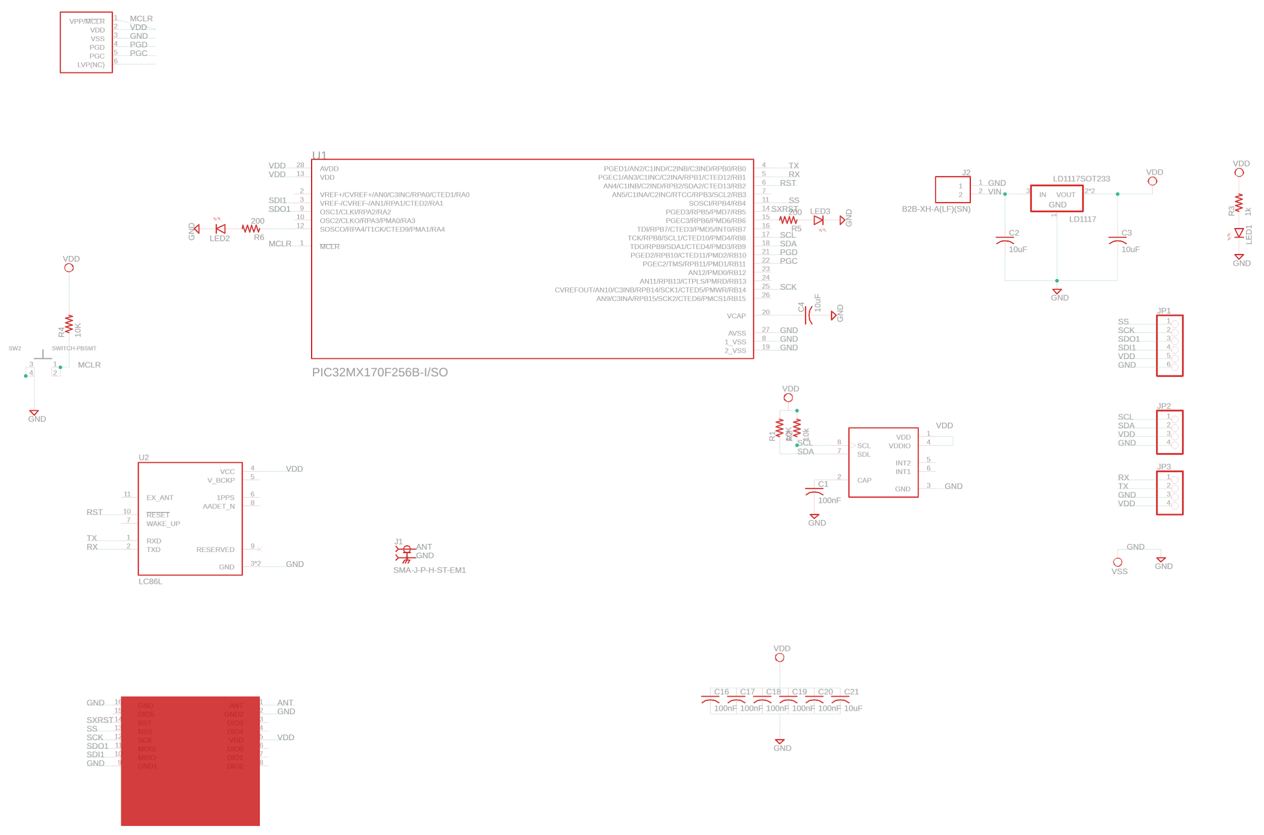
7 Conclusions

Overall this project was very challenging but very rewarding and enjoyable. There were many moving parts and that required coordination between all members of the team to ensure we were on the same page. The gulf of difficulty between understanding and implementing a system was far wider than we expected. This project did show that we are capable of getting the bare bones working and several of us are going to continue working on DroneHook over the summer and potentially longer.

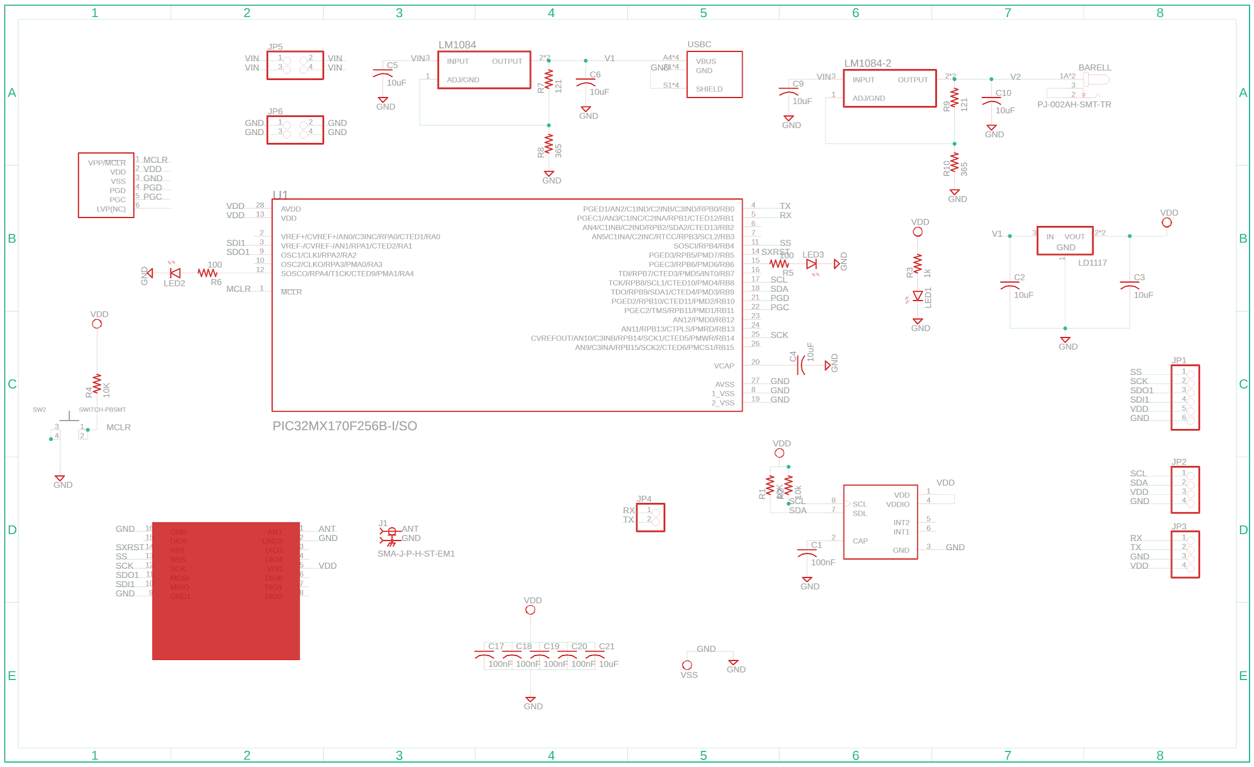
8 Appendices

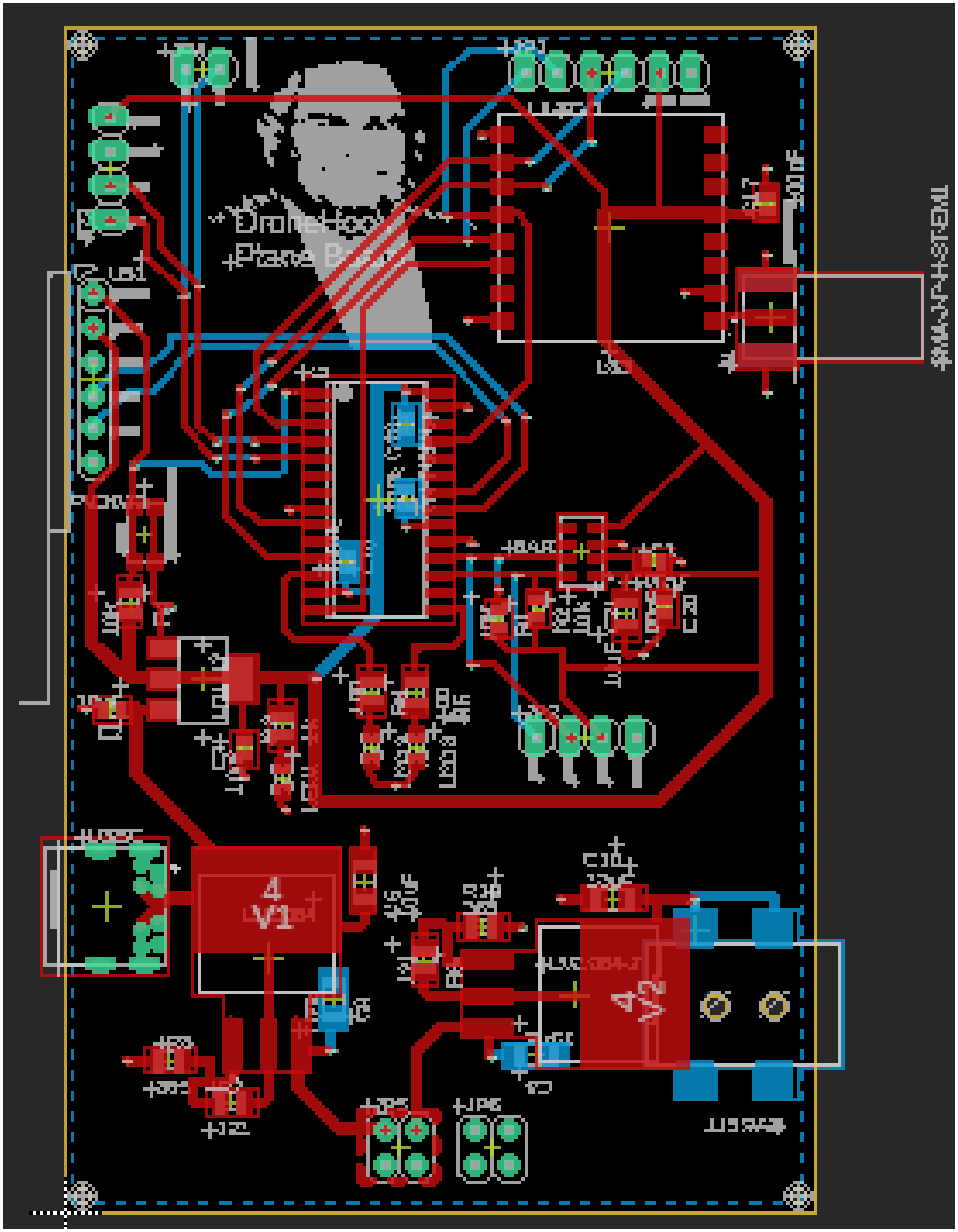
Eagle Schematics for On Balloon System




VPP/MCLR
VDD
VSS
PGD
PGC
LVP(NC)
MCLR
VDD
GND
PGD
PGC
U1
AVDD
VDD
MCLR
VDD
VDD
SDI1
SDO1
MCLR
TX
RX
RST
SS
SXRST
SCL
SDA
PGD
PGC
SCK
VCAP
AVSS
1_VSS
2_VSS
GND
PIC32MX170F256B-I/SO
LED2
R6
200
LED3
R5
C4
10uF
J2
B2B-XH-A(LF)(SN)
GND
VIN
LD1117SOT233
IN
VOUT
LD1117
C2
10uF
C3
10uF
VDD
R3
1k
LED1
JP1
JP2
JP3
SW2
SWITCH-PBSMT
R4
10k
U2
VCC
V_BCKP
EX_ANT
1PPS
AADET_N
RESET
WAKE_UP
RXD
TXD
RESERVED
LC86L
J1
ANT
SMA-J-P-H-ST-EM1
R1
R2
C1
100nF
INT2
INT1
CAP
VDDIO
C16
C17
C18
C19
C20
C21
100nF
10uF
VSS

Eagle Schematics for On Board System




DroneHook
Plane Board
V1
V2
LED3
RESET

Useful Links and Resources

Files:
Code, Eagle Files and CAD Files

Part Data Sheets:
PIC32 Microcontroller
LM1084
SX1276
LC86L
LD1117
Raspberry Pi 4