
# Final Report

*Prepared By:*
Richard Slocum

March 10, 2016

# Introduction

Cormorant Analytics improved upon a previous Multi-View Stereo Sensor by implementing a new prototype system consisting of new camera hardware, triggering electronics, 3D printed mount and enclosures, and Matlab processing algorithms. The sensor was successfully integrated on a 3D Robotics X8 Quadcopter UAS platform. A camera layout was selected to optimize the field of view of the sensor for long timeseries data acquisition of breaking waves in the littoral zone. Camera boresight and relative orientation between the cameras is calculated and reported using the CAD model. The system was succesfully integrated onto a 3D Robotics X8 Octocopter for initial testing. Cormorant Analytics also provided support for processing previously acquired imagery through Agisoft Photoscan, as well as generation of Orthophotos with previously developed algorithms. All of the Delivery Milestones, outlined in Table 1, were completed on time.

Table 1: All Milestones were completed on time.

| Delivery Order Milestones | Scheduled Completion | Actual Completion |
|---:|---:|---:|
| Project Kick-Off Meeting | 15-Mar | 15-Mar |
| Quarterly Program Management Review | 15-Jun | 15-Jun |
| Quarterly Program Management Review | 15-Sep | 15-Sep |
| Initial Prototype Delivery | 15-Oct | 15-Oct |
| Quarterly Program Management Review | 15-Dec | 15-Dec |
| Close-out Brief | 15-Mar | 15-Mar |

## Overview of Report

### Chapter 1: Camera Hardware

The decision to use GoPro cameras, as well as the alternative camera options considered are discussed. The lens and camera orientation are also discussed.

### Chapter 2: Electronics

The electronic components, schematic, board, and methodology for triggering via teh GoPro audio port are discussed. The cable pinout for each cable is also documented and discussed.

### Chapter 3: Mount Design and Fabrication

Each component for the sensor design is briefly discussed, as well as the calculation of the relative orientations of the cameras from the CAD model.

### Chapter 4: Image Synchronization

The methodology and algorithms used to synchronize the video files to UTC time is discussed. Documentation for how to extract images with interpolated GPS position info is also discussed.

## Chapter 5: Acquisition and Data Storage

A workflow and checklist for data acquisition is presented.

# Contents

# Chapter 1

# Camera Hardware

The previous version of the MVSS sensor could only attain a maximum synchronized frame rate of 0.5Hz, which is inadequate for celerity based bathymetry inversions. Alternative sensors were assessed and considered, with cost and weight being the major limitations. After consideration, an array of GoPro Hero 4 Black sensors with modified lenses were selected as the camera sensors.

## 1.1 Camera Selection

### 1.1.1 GoPro Hero 4 Black

GoPro cameras are a Commercial Off The Shelf(COTS) camera that is used by consumers across the world for video and imagery. While the GoPro sensors have a number of negatives, outlined in the advantages and disadvantages list in Table 1.1, they were ultimately selected due to the low weight, low cost, low power, availability for purchase, and ruggedness of the cameras.

| **Advantages** | **Disadvantages** |
|---|---|
| Inexpensive | CMOS rolling shutter (nonlinear image distortion) |
| Lightweight | No documented method for time synchronization |
| Easy to purchase | High lens distortion |
| Rugged | |
| Low Power Requirements | |

Table 1.1: Advantages and disadvantages of using GoPro cameras for use in the second version of the MVSS.

### 1.1.2 PtGrey

PtGrey is an industrial camera manufacturer that manufacturers a wide array of products which contain hardware triggering. While the hardware triggering of these sensors would be a much more robust and accurate time synchronization methodology, the weight and power requirements were too great for integration on a small Unmanned Aerial System. The PtGrey Ladybug was also considered, but the weight and power for it made integration on a UAS prohibitive. An advantages and disadvantages list for the PtGrey cameras is shown in Table 1.2.

| Advantages | Disadvantages |
|---|---|
| Hardware triggering | Too heavy with lenses and CPU |
| Well documented API | Requires external CPU for triggering and data storage |
| Reliable | |
| Many Lens Options | Requires a lot of power to run the camera and CPU |

Table 1.2: Advantages and disadvantages of using PtGrey cameras for use in the second version of the MVSS.

### 1.1.3 Raspberry Pi Camera Module

Raspberry Pi is an inexpensive, consumer grade linux computer that is marketed towards students and hobbyists. The Raspberry Pi computer interfaces with a small Raspberry Pi camera module, which can then be used to acquire imagery and video. After experimenting with the camera, it was determined that the software triggering accuracy of the camera was inadequate for time synchronization across an array of cameras. The other advantages and disadvantages are shown in Table 1.3.

| Advantages | Disadvantages |
|---|---|
| Inexpensive | Low resolution |
| Lightweight | Inaccurate software trigger |
| | Relatively "black box" camera module |
| | Requires a Raspberry Pi for each camera, adding to the power requirements and weight. |

Table 1.3: Advantages and disadvantages of using Raspberry Pi cameras for use in the second version of the MVSS.

### 1.1.4 CHDK

The Canon Hacker Development Kit(CHDK) is an open source project that modifies the firmware of Canon point and shoot cameras to enable advanced image acquisition. This was used for the first version of MVSS, but the frame rate was too slow. CHDK was again investigated to see if any improvements could be made to the frame rate, but it was determined that the frame rate could not be improved past the 0.5Hz.

| Advantages | Disadvantages |
|---|---|
| Accurate pseudo hardware trigger | Frequent Undocumented Bugs |
| High resolution (16Mp) | Poor Frame Rate |
| Lightweight | Retracting Lens causes unstable lens IO parameters |
| Low Power | |

Table 1.4: Advantages and disadvantages of using CHDK cameras for use in the second version of the MVSS.

## 1.2 Lens Selection

GoPro cameras are constructed with a 2.92mm lens by default, but there is an undocumented method to open the camera and replace the lens. The stock lens mounts into M12 threading, and the whole threading

mechanism can be removed and replaced with a product called a "SuperMount." This mount also has M12 threading, but allows for longer focal lengths to be installed. There were four lenses on the market that advertised a 'High Mp' lens for the M12 threading, and each are compared in Figure 1.1. The 2.9mm aftermarket lens was eliminated as an option because it exhibited excessive radial distortion and vignetting. The default, 4.35mm, and 5.4mm lenses were considered as options when designing the camera orientations.

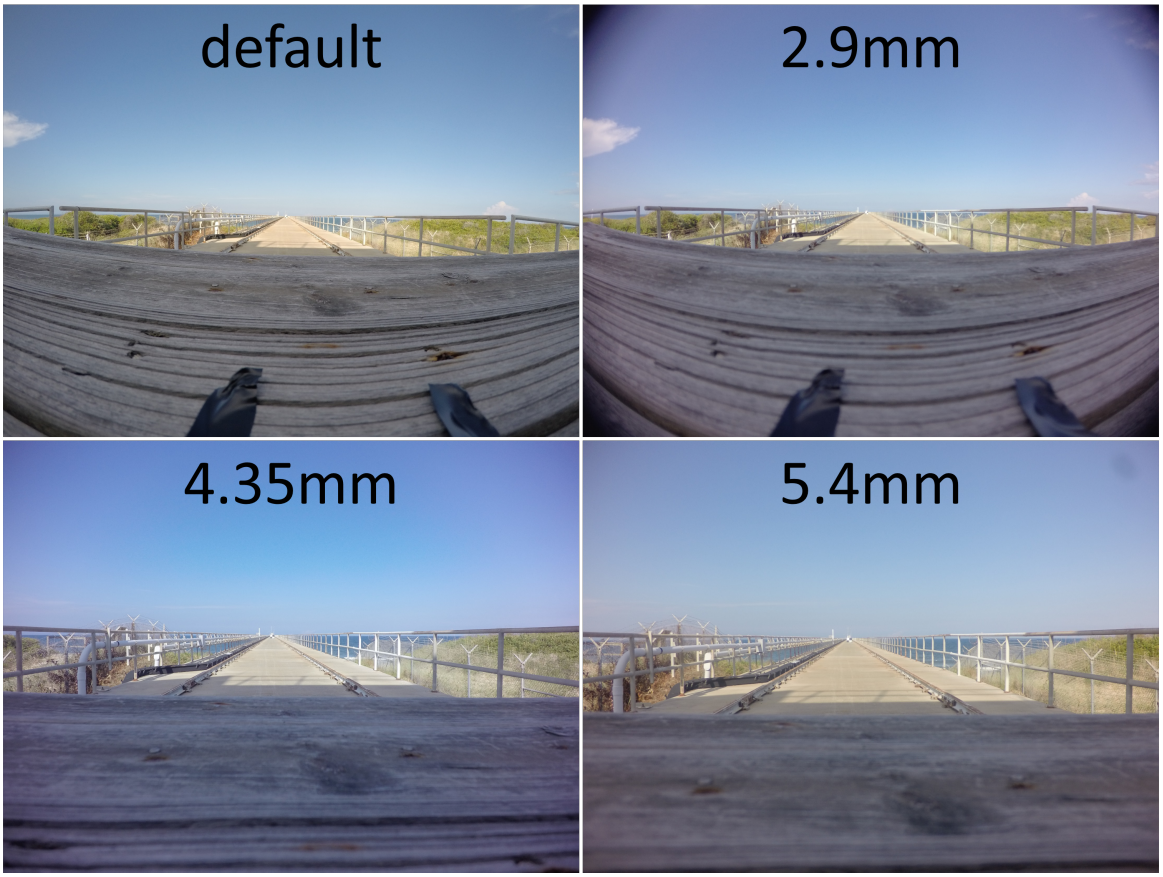



Figure 1.1: The four M12 lenses were tested in the GoPro camera, and it was determined that the 2.9mm lens should not be used due to excessive vignetting and distortion. The default, 4.35mm, and 5.4mm lenses were considered as options when designing the camera orientations.

## 1.3 Possible Improvements

The current GoPro sensor design functions as a prototype, but will likely require debugging and troubleshooting to become a robust sensor for repeated data acquisitions. The GoPro camera is the greatest limiting factor in the system, and efforts should be made to continue explore more robust, industrial camera options as camera technology progresses. Future work should continue to monitor the virtual reality camera and cell phone camera markets, as these consumer technologies hold great promise for an improved sensor. These types of cameras may have time synchronization built in, which is the largest issue with this current system.

# Chapter 2

# Electronics

Synchronization of the GoPro cameras is essential for seamless orthophoto generation and boresight calculation. The GoPro cameras do not have any documented methods to synchronize either video or imagery acquired with more than one camera. A custom solution was developed to utilize the external microphone port of the GoPros in order to embed metadata in any recorded video files. The approach using the audio port was selected, and is integrated in a modular manner, so that alternative video cameras with an external microphone option can be synchronized in the future using this same methodology.

## 2.1 Other Considered Methodology

An alternative approach using a GoPro remote trigger was also investigated, but proved to be inadequate. GoPro cameras can also be purchased with an external, handheld remote which operates over wifi to trigger one or more cameras. This approach was investigated and tested for feasibility by triggering four GoPro cameras indoors. The variable latency between when the remote was pressed and each GoPro began acquiring data was clearly discernable through the audible beep that each camera made when the command was received and acknowledged. The audible beep variability between cameras was sometimes on the order of tenths of a second, which is far too great of an offset for accurate triggering. Had the cameras been triggered at the exact same time, further investigation would have still been needed to ensure that there was no temporal drift between cameras.

## 2.2 Audio Encoding Technical Approach

The methodology to time sync the imagery relies on the two audio channels (left and right) of the GoPro. The left channel received the Binary counter integer once a second, and the right channel records the raw PPS from the GPS. The Teensy microcontroller records the exact time it sent the binary signal to the GoPro. This establishes a correspondence between the Teensy and GPS time frame. The GPS also sends the PPS and GPS NMEA string to the Teensy, which can be used to solve for a time synchronized pointcloud to UTC time.

The binary signals sent to the GoPro are timestamped with the GoPro internal time, which is synchronized to the video frames that are being acquired. These type of binary signals are normally decoded by a serial port, but a custom solution was developed to decode the information from the audio signal. These signals are then decoded in pos- processing to synchronize the GoPro Video frames to GPS time. This methodology only works when the GoPros are in video mode, and therefore the spatial resolution is limited compared to the image frame capture capabilities.

An initial approach to record raw NMEA sentences from a GPS receiver was unsuccessful due to difficulties in decoding the raw data, as well as bandwidth limitations. A theoretical maximum of only 48kbs could be sent over each microphone channel due to the fixed sampling frequency of the GoPro microphone port, however in practice this value is much lower due to signal conditioning. For this reason, an alternative approach utilizing a SD card for data logging was selected.

A microcontroller is used as the system master, and it interfaces with a GPS receiver, a SD card logger, and the GoPro external microphone ports. The microcontroller is constantly recording GPS NMEA data directly to the SD card, and logging it with an internal timestamp depicting when the GPS data was received. Note that this timestamp certainly has some latency associated with it, as the timestamp is only recorded once the GPS has processed the raw signals and the microcontroller has acknowledged receipt of the serial data. For this reason, the PPS signal is also recorded to both the SD card and the GoPro external right microphone channel. This PPS, when coupled with the NMEA string, can provide accurate timestamping with nanosecond precision. Due to the latency between video frames and audio recorded by the GoPro, as well as the interpolation between frames acquired at 30Hz, nanosecond precision can not be achieved with this methodology. Further details on the processing of the GoPro audio and Teensy logfile to achieve timestamped video is explained in detail in Chapter 4.

## 2.3 Component Selection

The components for the PCB were selected to minimize space, weight, and power requirements. The following components were selected and integrated onto the PCB, and a short description of the part is provided.

### 2.3.1 Teensy 3.2

`https://www.pjrc.com/teensy/teensy31.html`

The Teensy 3.2 microcontroller is a 32bit, breadboard friendly microcontroller that operates on 3.3V logic. It was selected mainly due to the small form factor and access to the three Serial ports required for interfacing with the SD card, GPS, and GoPro camera. Compared to Arduino brand microcontrollers with similar capabilities, the Teensy 3.2 is also lower cost and has increased performance. The Teensy can be programmed with the Arduino IDE, so the transition from Arduino code used in previous iterations of the sensor to the Teensy was seamless.

### 2.3.2 openLog

`https://www.sparkfun.com/products/9530`

The openLOG is a lightweight, easy to implement data logger that takes any serial data passed to the RX port and logs it to a microSD card. The SD card is initially set to a default 9600 baud rate, but the config.txt file on the sd card is modified for this project so that it logs at 115200. The SD card must have the config.txt file with the baud rate field set to 115200 for the current code to work.

### 2.3.3 Adafruit Ultimate GPS

`https://www.adafruit.com/products/746`

The Ultimate GPS is a low cost L1 only GPS receiver that has a PPS output. It was selected because it is low cost, lightweight and easy to integrate as it has the digital io pins broken out to headers. The PPS pin in conjunction with the NMEA GPS string provides accurate timestamping of the imagery, as well as a low quality GPS position.

### 2.3.4 Voltage Regulator

`https://www.adafruit.com/products/1065`

The “Mini DC-DC 5V Stepdown” regulator was chosen as it provides up to 94% efficiency and up to 1A output. It also has a 6.5-32V input range, which allows for various battery options to be explored. The high efficiency of the DC-DC stepdown enables smaller and lighter batteries to be used.

### 2.3.5 9 Degree of Freedom IMU

`https://www.sparkfun.com/products/13284`

The LSM9DS1 is a 9 degree of freedom motion sensing chip that houses a triad of accelerometers, gyroscopes, and magnetometers. This Chip was selected as it is very lightweight, low cost, and should contain the ability to generate low accuracy camera poses. It communicates via I2C protocol, and therefore an extra serial port is not required.

### 2.3.6 JST Connectors

`https://www.sparkfun.com/products/9916`

JST connectors are a brand of connector that connects a breadboard to a cable. These are purchased with the wires already broken out for ease of integration in 2-pin, 3-pin, and 4-pin connector packages.

### 2.3.7 "SuperBright" LEDs

`https://www.adafruit.com/products/754`

LED status lights were selected to be as bright as possible because previous data status LEDs were difficult to view in bright, outdoor lighting. These status lights use more battery, but as they are dubbed "superbright", they are much easier to see. The polarity of the pins of the LED are important, and are depicted by the length of the wire to the LED. The longer wire is the positive lead, and the shorter is the negative, as depicted in Figure 2.1.

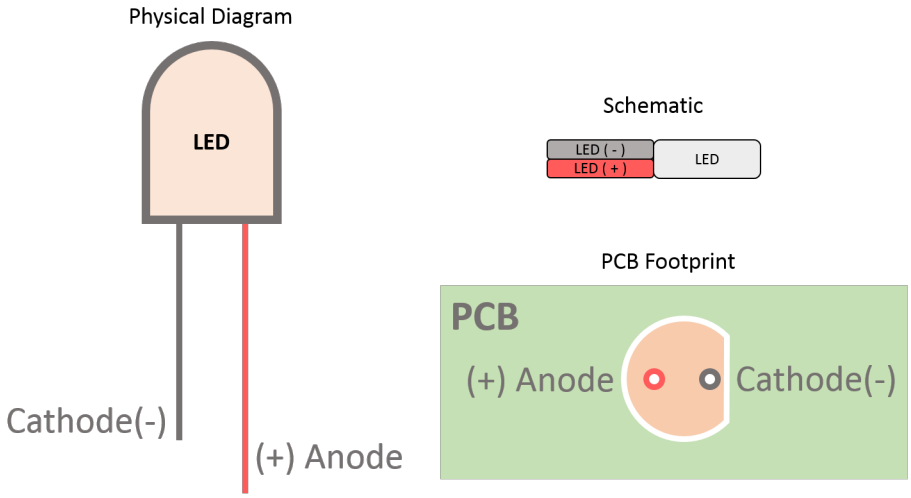



Figure 2.1: The LEDs used are follow a traditional LED diagram and PCB format, where the longer wire represents the Anode.

## 2.4 PCB Schematics and Board Designs

EAGLE was used to generate the schematic and board designs for the circuit. The design was divided into two PCBs. The first PCB, referred to as the "Main" PCB, contains all of the main components of the circuit. The second PCB, referred to as the "Panel" PCB, works as a connector so that the GoPro cables can be connected and disconnected easily from the outside.

### 2.4.1 Main PCB

The schematic for the main PCB, shown in Figure 2.2, is divided into the Power, Voltage Divider, Optional LEDs, Camera Output, Teensy microcontroller and Sensors sections. Notice that in the Voltage Divider section, in order to ensure the correct Voltage levels into the GoPro microphone port a voltage divider was used to reduce the 3.3V input to approximately 30mV for both the PPS and NMEA ports.

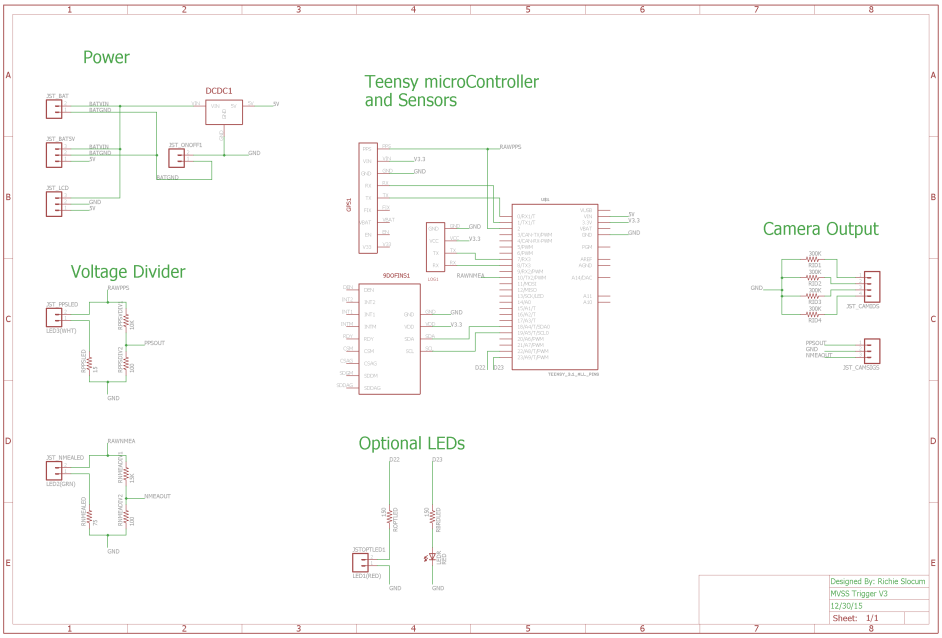



Figure 2.2: The main schematic was designed in EAGLE, and a higher resolution is available in the deliverables folder structure as both a ".sch" and ".png" file.

The main board layout, shown in Figure 2.3, was designed to minimize the footprint and reduce large cable bending by placing connectors near the edge of the board. The board was also designed in conjunction with the 3D printed mount, and the shape and orientation was optimized to fit on the electronics tray. The order that the components are soldered is important such that nothing covers up something that needs to be soldered from the other side of the board.

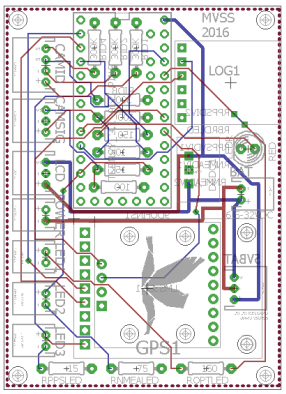



Figure 2.3: The main board layout was designed in EAGLE. Red traces represent the top of the board and blue traces represent the bottom.

### 2.4.2 Panel PCB

The Panel PCB serves as a connection point for the cables that go to the GoPro. The main PCB outputs the ground signal, two "pseudo-audio" signals, and an "ID" wire for each camera. The ground, as well as the PPS and binary "pseudo-audio" signals are split into 4 signals each so that the signal can be sent to each of the four cameras. The ID signals are individually sent to each camera, as the 300kΩ resistor tying the signal to ground is what tells the goPro to store audio from the "external audio" port. The schematic is shown in Figure 2.4, and the board is shown in Figure 2.5.

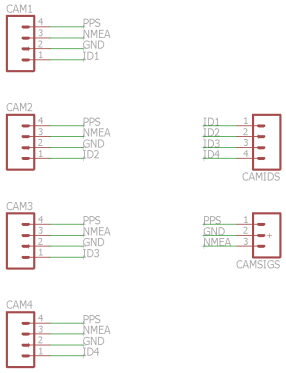



Figure 2.4: The panel schematic was designed in Eagle. This simple PCB is used as a connector so that the 7 signals are split to go to the 16 signals required for the GoPro.

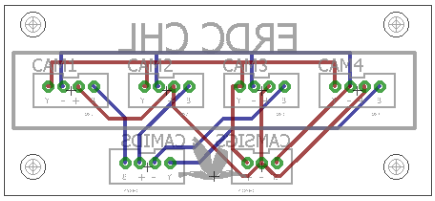



Figure 2.5: The panel board layout was designed in EAGLE. Red traces represent the top of the board and blue traces represent the bottom.

## 2.5 Cable Pin Labels

The pinouts for each of the two PCBs are depicted in Figure 2.7. These cables should be custom made to the correct length so as to reduce the overall weight of the system. Each of the JST connectors comes with the leads already crimped into the connector and tinned with solder on the other end. This allows for them to be easily soldered together to generate a solid connection.

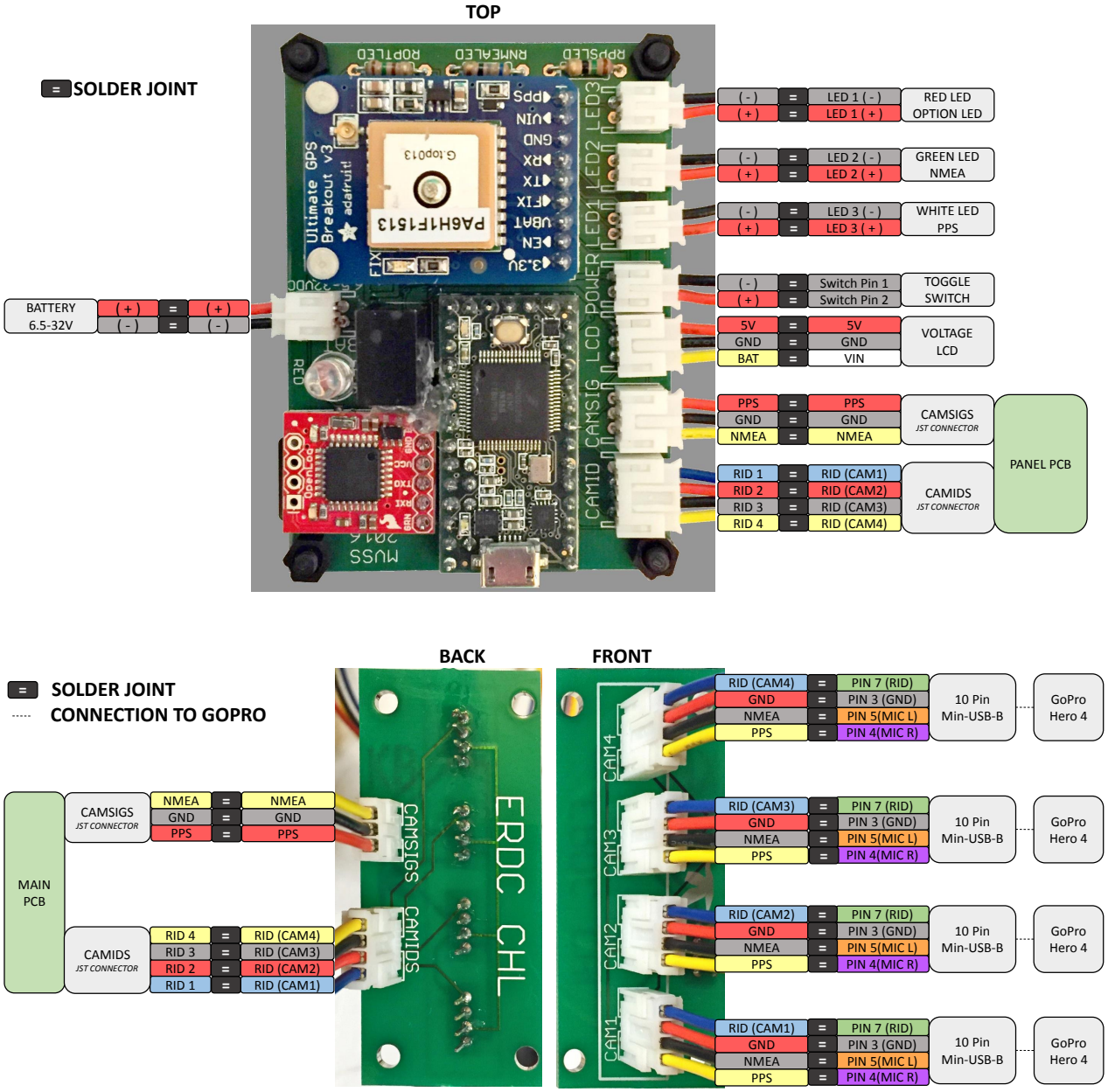



Figure 2.6: The pinouts for the main PCB and panel PCB are labeled. The color of the box represents the color of the wire.

## 2.6 GoPro Mini-USB-B Cable

The cable that connects the GoPro to the PCB is based on a USB mini-B cable with 10 pins. It utilizes the pins as shown in Table 2.1. The 300kΩ resistor could have been placed within the cable, or immediately after the socket, as both channel 7 and 3 are right next to each other and a resistor could simple be soldered across the two. This would reduce the number of wires in the cable from 4 to 3. However for this version of the cable the resistor is placed on the PCB rather than in the cable to improve the ability to debug the

system.

| Pin Number | Description |
|---|---|
| Pin 3 | Ground |
| Pin 4 | Right Audio Channel |
| Pin 5 | Left Audio Channel |
| Pin 7 | ID pin which enables the external audio when brought to ground across a 300kΩ resistor. |

Table 2.1: The GoPro cable utlizes pins 3,4,5, and 7 to trigger the camera via the audio port.

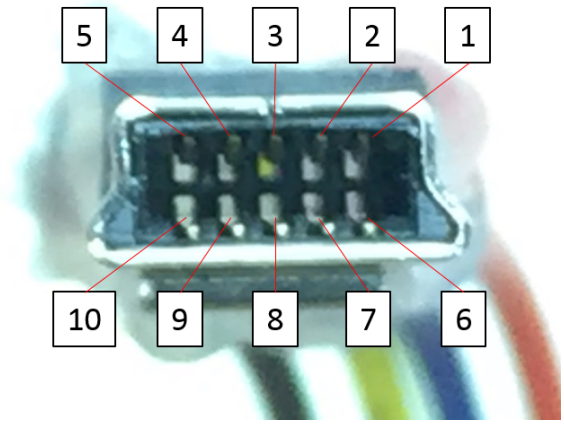



Figure 2.7: Traditional cables that are purchased in retail stores only break out 4 of the 10 pins in a mini-B connector. The GoPro triggering cable requires a custom cable, as pins 3,4,5 and 7 are used.

## 2.7 Power

The power input into the PCB is set up for 6.5V - 32V DC by default, with a maximum output of 1A at 5V. The average current draw was tested for a 9V battery. The average power draw is calculated using Equation 2.1 shown in Table 2.2, and estimated runtime calculated with with Equation 2.2 different batteries is shown in Table 2.3.

$$Average\ \ Power = Voltage \times Current \tag{2.1}$$

| Average Voltage (V) | Average Current (mA) | Average Power (W) |
|---|---|---|
| 8.3 | 0.080 | 0.664 |

Table 2.2: The average power draw for the PCB was calculated using a 9V battery.

$$Estimated\ \ Runtime(hours) = \frac{Battery\ \ Voltage \times Battery\ \ Current}{Average\ \ Power} \tag{2.2}$$

| Battery Type | Voltage | Estimated Capacity(mah) | Estimated runtime(hours) |
|---|---|---|---|
| 9V Battery | 9 | 400 | 5.4 |
| Two 1000mA Lithium Ion | 7.4 | 1000 | 11.1 |
| Three 1000mA Lithium Ion | 11.1 | 1000 | 16.7 |

Table 2.3: The estimated runtime is calculated for a number of potential battery options.

### 2.7.1 Alternate Power

An alternate method for powering the PCB is provided via a 3-Pin JST connector on the bottom of the board. The pinout for this port is not documented, as it has not been tested. The board and schematic should be read to determine the proper power input values. The connector provides the user access to provide their own regulated 5V battery circuit. There is one wire for ground, one for 5V, and another for raw voltage. The raw voltage line is used to monitor battery status and health using the LCD on the front of the panel.

## 2.8 Microcontroller Algorithm

The algorithm to write the logfile to the SD card is programmed using the Arduino IDE. The Arduino IDE is used as the programming environment for the Teensy, per documentation on the Teensy website. A flowchart depicting the code is shown in Figure 2.8, and the code is provided in electronic form. Emphasis is placed on ensuring accurate timing, but precedence is placed in this order:

1. Accurate retrieval of Teensy time when PPS rising edge is detected
2. Accurate Teensy time of when counter was sent to gopro
3. Accurate Teensy time of when the GPS time is received

The three different time references that are used for time syncing are GoPro time, Teensy time, and GPS time. GoPro time is the internal time of the GoPro camera in seconds from the time the video began recording. Teensy time is the time in milliseconds since the Teensy was powered on. GPS time is the Coordinated Universal Time(UTC), which is output from the GPS. The synchronization between these times is described in more detail in Chapter 4, but it is essential that the times are recorded as precisely as possible.

To ensure that the PPS and Teensy time are accurately correlated, an interrupt routine is implemented in the Teensy algorithm. An interrupt routine, while technically different than running in parallel, can essentially be seen as a parallel processing task. In this algorithm, the interrupt is constantly monitoring for the rising edge of the PPS pulse. When the rising PPS pulse is detected, the code stops whatever it is doing and jumps directly into the interrupt routine. The interrupt records the exact Teensy time when this occured, and saves it to a global variable before releasing the code to finish performing whatever task it was working on. It then writes the data to the SD card whenever it is convenient, as once the exact Teensy time is recorded there is no urgency to write the data.

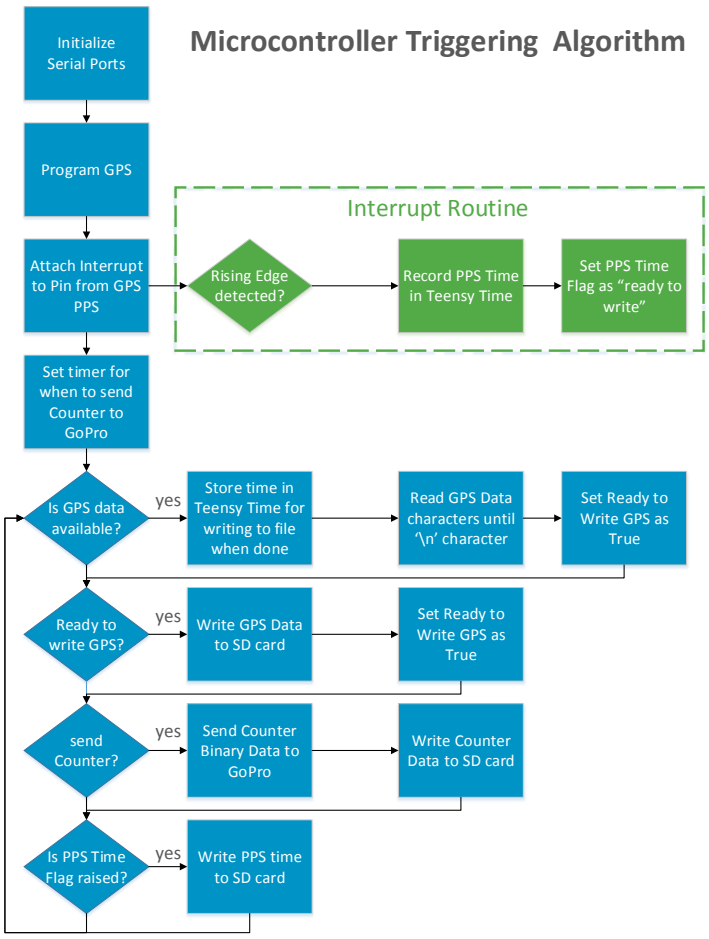



Figure 2.8: The Teensy algorithm is written to attempt to ensure as accurate timing synchronization as possible.

The output log file is structured similarly to the NMEA format, but does not enact checksums for each line. Each line of data is sent as a comma delimited line, where the first column is the address field, or identifier preceded by a '$'. The second column in each line is always the time in milliseconds from the time the microcontroller was turned on. The relevant id fields are described in Table 2.4. An example snippet of data from the file is shown in Figure 2.9. A script for parsing the data is shown in Algorithm C.4.

| ID | Description |
|---|---|
| $MSG | Any Raw GPS NMEA Strings |
| $IND | Counter number that was sent to the GoPro |
| $PPS | No Info, besides the time it was received at |

Table 2.4: The ID for each log file line is essential to parsing the data.

```
4155  $MSG,652387,$GPVTG,144.29,T,,M,0.15,N,0.28,K,A*39CRLF
4156  $IND,652501,652CRLF
4157  $PPS,652797CRLF
4158  $MSG,653209,$GPGGA,121110.000,4433.6010,N,12316.2727,W,1,09,0.93,62.9,M,-21.0,M,,*6CCRLF
4159  $MSG,653288,$GPGSA,A,3,20,21,18,15,13,10,29,27,26,,,,1.24,0.93,0.82*00CRLF
4160  $MSG,653350,$GPGSV,3,1,10,18,79,227,18,21,78,037,23,15,44,087,26,10,40,233,23*73CRLF
4161  $MSG,653424,$GPGSV,3,2,10,20,36,057,23,27,27,313,34,29,23,150,17,13,22,047,30*71CRLF
4162  $IND,653501,653CRLF
4163  $MSG,653499,$GPGSV,3,3,10,26,21,242,22,40,,,*4FCRLF
4164  $MSG,653534,$GPRMC,121110.000,A,4433.6010,N,12316.2727,W,0.04,144.29,040316,,,A*70CRLF
4165  $MSG,653613,$GPVTG,144.29,T,,M,0.04,N,0.08,K,A*3BCRLF
4166  $PPS,653797CRLF
4167  $MSG,654247,$GPGGA,121111.000,4433.6010,N,12316.2727,W,1,09,0.93,62.9,M,-21.0,M,,*6DCRLF
4168  $MSG,654327,$GPGSA,A,3,20,21,18,15,13,10,29,27,26,,,,1.24,0.93,0.82*00CRLF
4169  $MSG,654389,$GPRMC,121111.000,A,4433.6010,N,12316.2727,W,0.02,144.29,040316,,,A*77CRLF
4170  $MSG,654466,$GPVTG,144.29,T,,M,0.02,N,0.03,K,A*36CRLF
4171  $IND,654501,654CRLF
4172  $PPS,654797CRLF
4173  $MSG,655221,$GPGGA,121112.000,4433.6010,N,12316.2727,W,1,09,0.93,62.9,M,-21.0,M,,*6ECRLF
4174  $MSG,655301,$GPGSA,A,3,20,21,18,15,13,10,29,27,26,,,,1.24,0.93,0.82*00CRLF
```

Figure 2.9: The Teensy log file is recorded similarly to a NMEA structure, with each line beginning with an ID preceded by a '$' symbol, and followed by the time in Teensy time.

## 2.9 Potential Electronics Improvements

### 2.9.1 Fix Blunders in PCB Board

There are a few improvements that could be made to the PCB and electronics for future versions. The first improvement is an improved main PCB. The PCB design used for this version had a few minor errors. These error included: the DC-DC having to be mounted backwards, a slight offset of the IMU pins, and a few incorrect resistor values. These fixes were manually implemented when the PCB was soldered together, and also electronically implemented in the board and schematic diagrams. The new files are saved and have been delivered electronically.

### 2.9.2 Integrate onboard IMU

The IMU is soldered to the board and connected to the SDA and SCL ports, however it is currently unrecorded. The initial focus was to ensure accurate timestamping before adding in more read write cycles from the IMU. While this IMU is very low cost, and low accuracy, it will most likely be able to provide a rough estimate of camera orientation.

### 2.9.3 Improve GPS Quality

A large improvement would be to improve the quality of the GPS receiver. The current L1 only GPS is accurate to less than 3m per the documentation, and does not record raw pseudo-ranges for post-processing. Integration of a RTK GPS, L1 and L2 GPS, post-processing enabled GPS, or a fully integrated INS would greatly benefit the calculation of the camera pose and orientation. The Ultimate GPS is placed into a socket on the PCB, with the thought that in the future an improved GPS sending NMEA strings could be integrated into the workflow with possibly only software modifications.

### 2.9.4 Implement Charge Port and USB Hub

A final improvement could be implemented to improve the usability of the system by adding a charge port and a USB hub to enable charging and reading from the sensors via one cable. The current workflow for getting data out of the system involves removing the SD card or plugging a separate USB cable into the USB port. An integrated switch on the Panel PCB could be a good place for a USB hub or a battery charging circuit.

# Chapter 3

# Mount Design and Fabrication

A mount was designed and fabricated to house the cameras and triggering electronics so that it could be mounted to a X8 UAV platform. The constraints and goals of the design were as follows:

- Weigh under 800g (X8 payload limit)
- House 4 GoPro cameras and triggering electronics
- Rigid, to ensure constant relative orientations between the cameras
- Easily manufactured for rapid prototyping and improvements
- Designed so cameras and electronics can be easily removed
- Little to no tools required
- Low cost

## 3.1 3D CAD Design

Given the design constraints, it was clear that 3D printing using PLA or ABS plastic provided the most robust manufacturing methodology. The plastic is lightweight, easily manufactured, perfect for rapid prototyping, low cost, rigid, and can be designed with intricate interlocking parts to minimize the use of tools. Each part of the sensor that was 3D printed was designed for 3D printing to minimize the amount of support required. Each part has an optimal orientation that it should be printed at, so as to maximize structural integrity and minimize support. For example, a five sided hollow cube should be printed so that the open face is up, so that no support is necessary. The orientation of each of the parts should be intuitive to the user.

### 3.1.1 Camera Enclosure Design

The GoPro camera enclosure, shown in Figure 3.1 was designed so that the cameras could be easily removed, as well as fit tightly into the enclosure so that there is no movement or shifting during the flight. Many prototypes were generated, and various tolerances were investigated to ensure a compromise of "easy to remove" and "doesn't move while in flight." A hinged design was selected as it minimized the number of screws or tools required, and was also a simple design to 3D print. The lens protrudes into the threaded cylinder on the lid, and all of the ports and buttons are accessible. The enclosure consists of 3 parts:

1. Enclosure (yellow): This is the main body that the GoPro mounts into
2. Lid (red): This is the hinged part that locks the camera into place
3. Cap (not shown): This 3D printed part threads onto the lid to protect the GoPro lens when the cameras is not acquiring imagery.

The lid and cap are printed individually, while the enclosure is mounted to the Camera Mount in software, and printed as a fused, solid part, as shown in Figure 3.6.

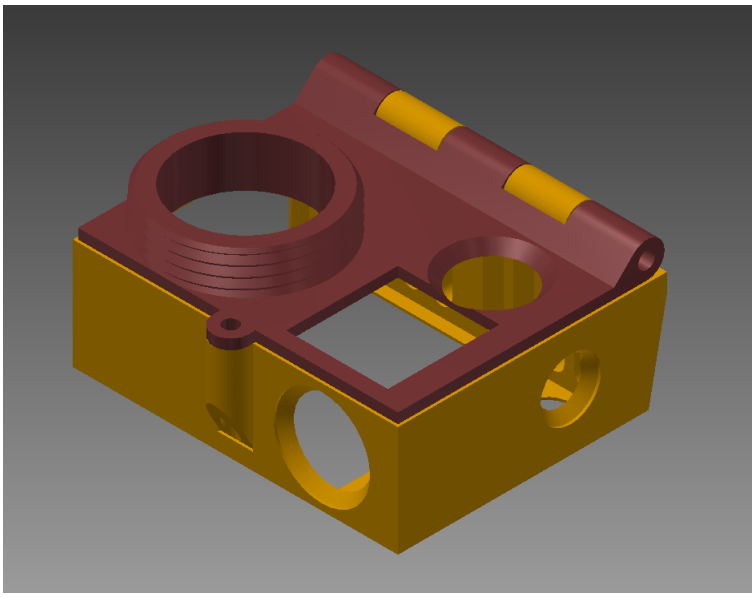


Figure 3.1: The camera enclosure was designed with a hinge, and one locking screw. The threaded area around the cylinder for the lens enables a protective cap to be screwed on.

### 3.1.2 Camera Mount Design

The camera mount, shown in Figure 3.2, defines the orientation of each of the cameras, and is the main frame of the design. The camera enclosures are mounted directly to each of the four bottom planes in software, and together they are printed as one piece. This print requires some support material to be removed, but ensures less tools and moving parts in the final design. The proper orientation for printing the Camera Mount is with the wide opening for the electronics tray facing upwards, so that no support material is required within the body.

The camera mount is designed so that the electronics tray can slide in and out such that the triggering electronics may be removed easily. It also incorporates a honeycomb pattern on the top to reduce the overall weight of the system. The four legs are designed so that vibration damping balls may be placed to connect the system to the X8 Quadcopter legs, shown in Figure 3.4.

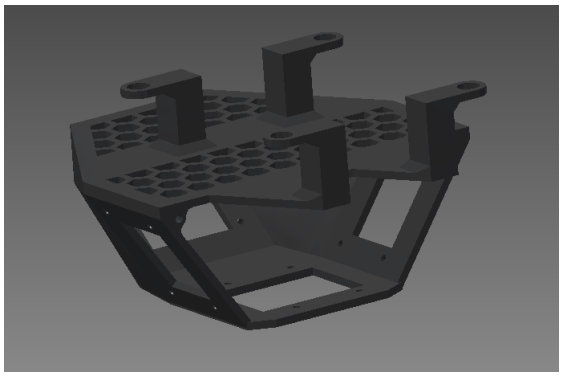


Figure 3.2: The camera mount was designed separately from the enclosure, so that the 4 cameras could be mounted to the design in software and they could be printed as one part.

### 3.1.3 Electronics Tray and Panel

The electronics tray, shown in Figure 3.3, was designed for easy access to the electronics PCB, while still providing status lights and a power switch on the outside. The tray is locked in with two screws in the upper corners, to ensure it does not slide out during a flight. The panel portion of the tray, houses three status LEDs, a Voltage LCD display, a power switch, the panel PCB for connections, and an external antenna connection point. The decoupling of the electronics tray from the mount allows for various electronics packages and batteries to be integrated into the system, without having to redesign a new mount.

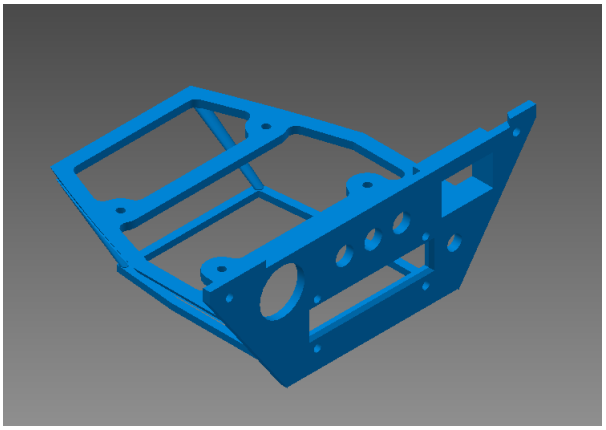


Figure 3.3: The electronics tray was designed so that the PCBs could easily be slid in and out, while the panel provided power and status lights.

### 3.1.4 X8 Quadcopter Legs Design

The X8 legs mount to the X8 chassis using three bolts. These straddle the battery, and provide a socket for the vibration damping balls to be attached to. These vibration damping balls, shown in Figure 3.5, are integrated to reduce the propagation of vibration from the motors to the cameras. Both 200g and 300g vibration damping balls were provided, as the optimum type has not been determined. Based on the weight of the system, four 200g vibration damping balls should provide a good initial amount of vibration damping.

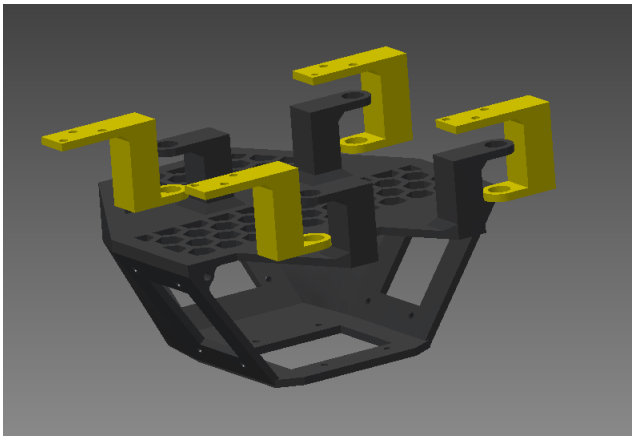


Figure 3.4: The X8 legs were designed so that a vibration damping ball could be placed in between the legs to reduce vibration.

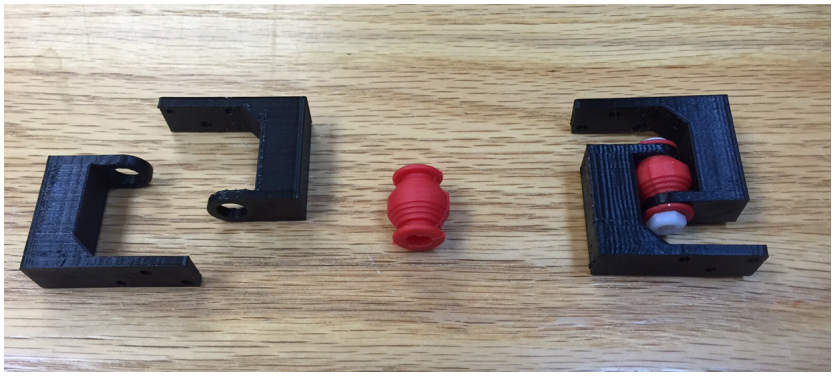


Figure 3.5: The vibration damping balls, shown in red, are used to connect the X8 to the mount and reduce the vibration from the X8 motors.

## 3.2 Full Design Specs

The final system weighs  700g with all of the cameras and electronics on board, and can be printed with one 1kg spool of printer filament. The full system, shown in Figure 3.6 is mounted to the X8 and can acquire imagery for approximately 15 minutes. The fully mounted system is shown in Figure 3.7.

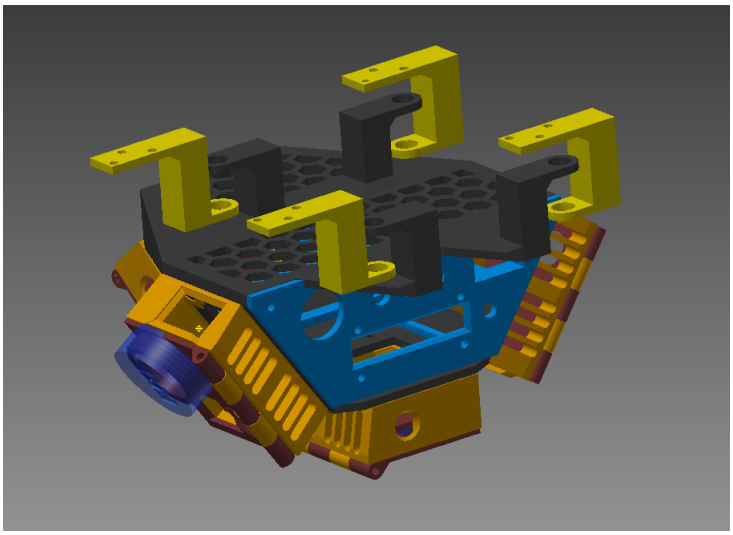


Figure 3.6: The camera mount was designed separately from the enclosure, so that the 4 cameras could be mounted to the design in software and they could be printed as one part.

Figure 3.7: The sensor is mounted to the bottom of the X8 with the GoPros installed for a proof of concept flight.

## 3.3 Camera Orientations

One of the main benefits of the 3D printed CAD Design, is the relative orientations between the cameras can be measured in software. Although there are sure to be uncertainties with printing and camera placement, the measured orientations should be very accurate. In order to define the coordinate systems, a central 'Mount coordinate system' is defined to align with the X8 body at approximately the center of the mount, as shown in Figure 3.8. The camera is defined such that the focal point of the camera is the origin, Z points outward from the camera, Y is down when the camera is facing upright, and X is along the length of the camera to the right. The transformation from each camera to the mount is defined in a right handed coordinate system based on the Tait Bryant Euler angles. The order of rotations is Z($\Psi$)-Y($\Theta$)-X($\Phi$), and the rotation is defined in Equation 3.1, where DCM represents the Direction Cosine Matrix.

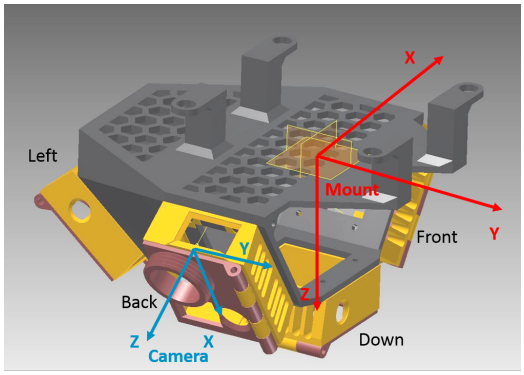



Figure 3.8: The camera coordinate systems defined from the CAD model are relative to the mount coordinate system, which is defined as an arbitrary point on the top center of the mount.

Table 3.1: The rotation and translation from the camera coordinates to the mount coordinates is defined in a right hand coordinate system where Yaw, Pitch, and Roll are defined as right hand rotations about the Z,Y, and X axes respectively.

| Camera Name | Lens (mm) | Yaw($\Psi$) | Pitch($\Theta$) | Roll($\Phi$) | X (in) | Y (in) | Z (in) |
|---|---|---|---|---|---|---|---|
| Back | 2.9 | 0° | 55° | 0° | 2.74 | 0.904 | -0.931 |
| Down | 5.4 | -90° | 0° | 0° | -0.404 | 1.014 | -2.587 |
| Front | 2.9 | 0° | -55° | 0° | -2.085 | 0.904 | -1.867 |
| Left | 2.9 | 180° | 0° | 40° | -0.571 | 3.558 | -1.92 |

$$X_{mount} = R \times X_{camera} + T \tag{3.1}$$

$$R = DCM(\Psi, \Theta, \Phi) \quad T = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

## 3.4 Possible Improvements

### 3.4.1 Reduce Weight

The current sensor design weights about 700g, but not much optimization in the design for weight reduction has been performed. If there is a large potential benefit that could be gained from a reduction in weight, the current mount and designs have some room to remove material while maintaining rigidity. An alternative method, while more intensive, would be to incorporate carbon fiber or some other material onto the 3D printed parts to increase rigidity.

### 3.4.2 Implement Weatherproofing

Sand or moisture can currently get into the electronics enclosure through the honeycomb pattern on the top of the mount. If this becomes a noticeable issue, the mount should be redesigned to seal off this opening. This could be achieved with something as simple as tape across the gaps, or by adding a thin layer in the CAD design. If the sensor is to be used in harsher environments, it could be redesigned with gaskets and more robust sealing mechanism to become more fully weatherproof.

### 3.4.3 Improve Panel

The first version of the panel contains various lights and LEDs to keep the user knowledgeable on the status of the sensor. This panel could be modified to either add or remove status leds, or even a small LCD screen for text status updates. If the status lights and LCDs are determined to not be useful, 'improvement' could actually mean removing some components.

### 3.4.4 Improve Vibration Damping

The vibration damping implemented is a first cut, low cost approach to attempt to remove the vibration from the system. If this method is not adequate, further investigation into vibration damping systems will need to be performed. The design of the mount is modular enough that the only changes would be to the Legs of both the X8 bracket and the mount.

# Chapter 4

# Image Synchronization

The cameras are each synchronized to UTC time using the audio data and PCB log file. Once each camera is synchronized to UTC time, frames are extracted at regular intervals for each video, and these images are said to be "synchronized." These videos are not however, hardware triggered and therefore the shutters are not synchronized. The theoretical accuracy of this methodology results in the cameras synchronized to a worst case scenario of 1/30s when ignoring all other sources of noise.

## 4.1 Technical Approach

The algorithm to develop the synchronization of the cameras from GoPro time to UTC time was developped in Matlab. The logfile, as described in Section 2.8, records the GPS data, PPS rising edge time, and the time the binary count value was sent to the GoPro. The GoPro audio channels recorded the PPS signal and the binary count value from the Teensy microcontroller. Each of these data has redundant observations between the time frames, which enable us to perform a least squares adjustment between each of the time frames. When the processing is complete, p1 and p2 coefficient values as described in Equation 4.1, are recorded in a text file.

$$t_{UTC} = p1 \times t_{gopro} + p2 \tag{4.1}$$

The processing is broken up into four main algorithms:

1. Calculate GoPro time to UTC time coefficients
2. Extract Frames from GoPro Video at UTC times
3. Convert logfile into a CSV file with GPS info and UTC timestamps
4. Interpolate GPS position info for each image based on corresponding UTC times

### 4.1.1 GoPro to UTC time

#### GoPro Audio Signal Decoding

One challenge with extracting the binary and PPS data from the audio channel of the GoPro was filtering out the GoPro signal conditioning of the input signal. For example, Figure 4.1 demonstrates how a square PPS signal appears in the recorded audio file. Notice how the PPS signal, which should be a square wave, actually decays to 0 despite the voltage. Therefore, an algorithm was written to detect large changes in signal, which then represent changes in voltage. For the PPS signal, the detection of rising and falling edges of the PPS pulse was trivial, but it became slightly more complicated for the binary signal.

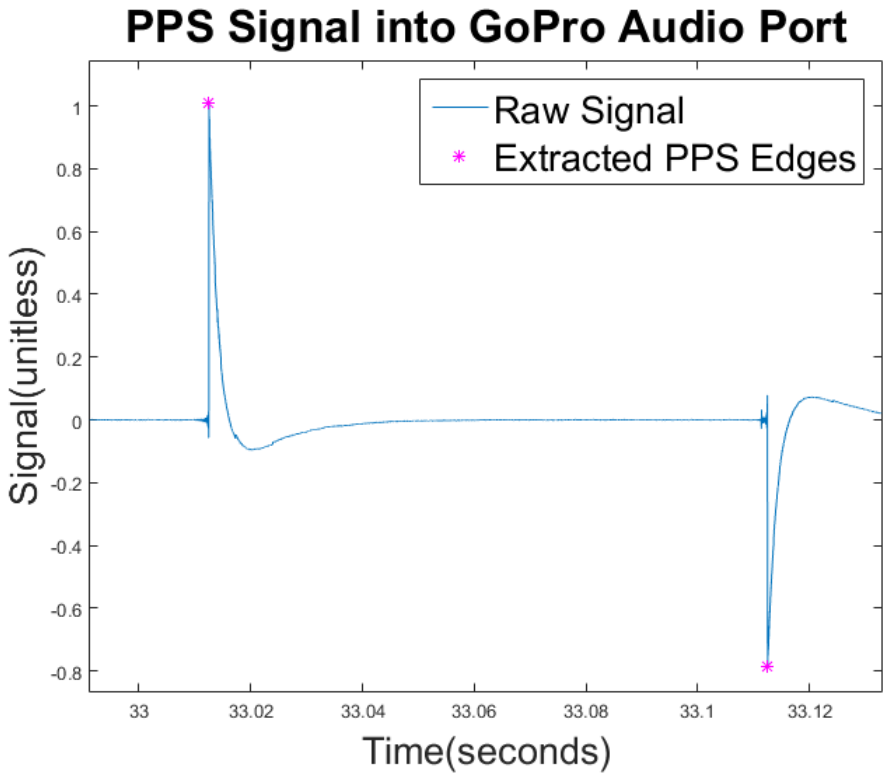



Figure 4.1: The PPS, which should be represented as a square wave, actually decays to 0 regardless of the voltage. The algorithm therefore is written to detect the large gradients, which represent a change in voltage.

The binary signal is set to 300 baud in the Teensy algorithm however it actually is sent at 1660 baud. This is acceptable, as the peaks were still discernable at this baud rate. If however, a faster baud rate were used, the peaks would become too close together and be difficult to decode. Figure 4.2 is an example decoded binary signal. The data is sent at 8N1 with no parity, so for the 2 byte binary integer being sent, it uses 20 bits. The raw signal is filtered to detect large changes, then a Gaussian filter is applied to smooth out the data. The peaks are then extracted at the known baud rate, with a peak threshold set as 1/3 the absolute value of the max peak value. The signal is then decoded into a binary signal, using an algorithm that takes into account that the signal will only change when the bit is changing. For example, notice how the 17th bit is high, and the 18th bit is 0, but both those values represent a 1 in binary. The bit value does not change to 0 until a low peak is detected at bit 19.

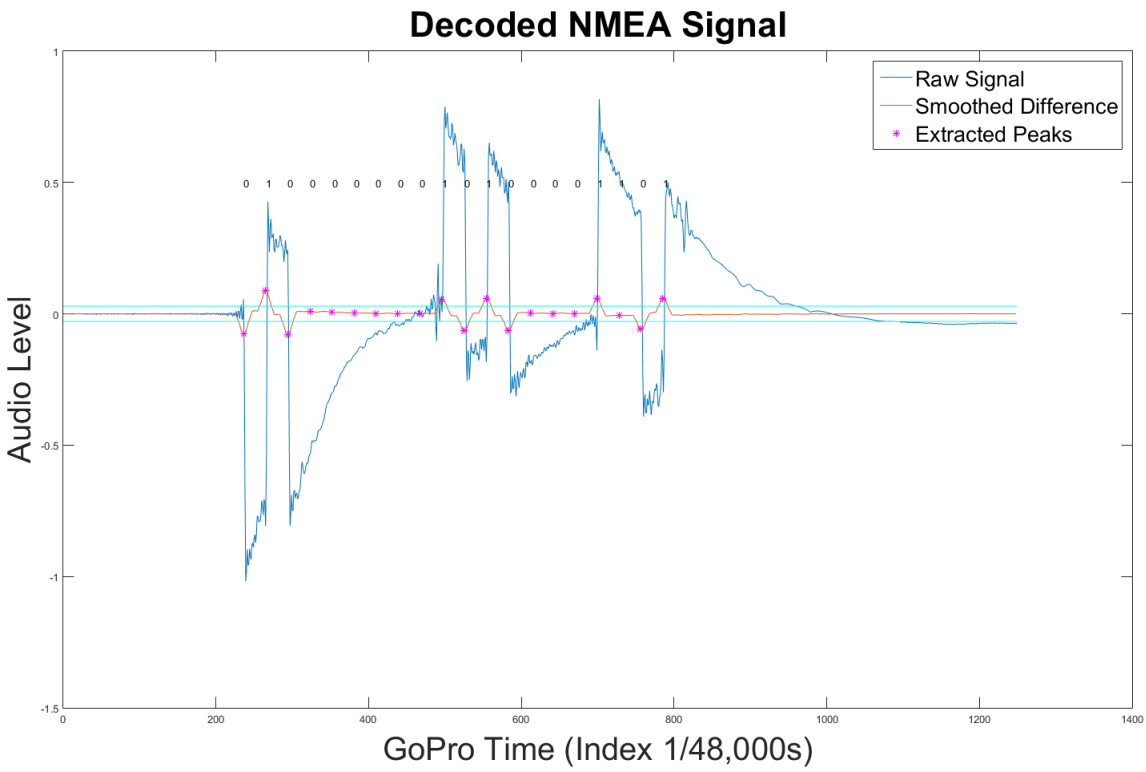



Figure 4.2: The binary signal sent to the audio port suffers the same decay as shown in the PPS signal. Here, the change in signal was used to detect when a change in bit value occurred.

### Algorithm Pseudocode

The conversion from GoPro time to UTC time can be performed without a PPS signal, but the absolute quality of this conversion will suffer due to latency in the Serial NMEA port and buffering. However, the relative synchronization between the cameras should remain the same, as any latency in reading the Serial port will be constant across each camera. The different time frames and data exchange between the Teensy, GoPro, and GPS PPS are shown in Table 4.1. Note that there is an ambiguity when attempting to correlate two PPS signals, as the PPS signal simply refers to a start of a second. If two PPS triggers are detected in two different time frames, there is no direct way to correlate the two. For this reason, GPS data sends NMEA strings to put the PPS second into an absolute UTC time frame. The function to calculate this offset and save a text file has been electronically delivered, and is also shown in C.2.

Table 4.1: The different data exchanges are recorded in three different time frames. Note that the X represents a direct correspondence between the data, while a '*' represents an ambiguity in the correspondence. The PPS values are sent at the beginning of each second in time, but there is no context to what actual second it is.

| Time Frame | Index | GPS NMEA | GPS PPS |
|---|---|---|---|
| GoPro | X | | * |
| Teensy | X | X | X |
| UTC | | X | * |

The steps to achieve time synchronization between GoPro time and UTC time are outlined below:

1. Extract All data from the SD card and the GoPro Audio

2. Use correspondences between Teensy Time and GoPro time for the index counter sent to the audio channel to calculate the conversion between the two time frames.
3. Use the NMEA GPS string recorded in Teensy time and stamped with UTC time to calculate a conversion between Teensy time and UTC time. *NOTE, if no PPS is detected, the algorithm stops here
4. A PPS pulse is described as having a rising edge exactly on the second with precision to the nanosecond. Using this knowledge, and the PPS timestamps in Teensy Time, estimate each PPS time in UTC time. Round the value to the nearest second after using the Teensy to UTC conversion
5. Convert the GoPro PPS times to UTC time, and round to the nearest PPS in UTC time. This assumes that the time is accurate to less than 0.5 seconds, so that the rounding does not become out of phase.
6. Calculate a new conversion from GoPro time to UTC time using the PPS correspondences.
7. Save this information to a text file '*_gopro2gps.txt.'

### 4.1.2 Extract Frames

Frames are extracted from the video imagery using the known conversion from GoPro time to UTC time, and a user-defined vector containing the times to extract imagery at. The algorithm to extract these frames and save imagery and metadata is provided electronically and also located in C.3. The pseudocode for the algorithm to extract frames is as follows:

1. Read the MP4 Video file into Matlab using Matlab's *VideoReader* Class
2. Read the previously calculated goPro to UTC time constants
3. Using the user input vector of desired UTC times for frames, calculate the correct frame indices to extract imagery
4. Write each frame to a JPG in a user defined folder, and save an 'iminfo.txt' CSV file which contains 3 Columns. In order to quantify the timing errors, the actual and desired times are both written. An example output file is shown in Figure 4.3
    - Image Name
    - Image Actual Time (This is the actual time of the frame)
    - Image Desired Time (This is the time the user wanted the frame at)

```
1 ImageName,ActualTime(yyyymmddHHMMss.fff),DesiredTime(yyyymmddHHMMss.fff)CRLF
2 00089_GOPR6864.jpg,20160304120755.989,20160304120756.000CRLF
3 00090_GOPR6864.jpg,20160304120757.991,20160304120758.000CRLF
4 00091_GOPR6864.jpg,20160304120759.993,20160304120800.000CRLF
5 00092_GOPR6864.jpg,20160304120801.995,20160304120802.000CRLF
6 00093_GOPR6864.jpg,20160304120803.997,20160304120804.000CRLF
7 00094_GOPR6864.jpg,20160304120805.999,20160304120806.000CRLF
```

Figure 4.3: The image info text file is comma delimited and is used to store metadata about the extract video frames.

One issue that arose is that Windows machines running operating systems that predate Windows 10 do not contain the codecs for reading 4K video by default. Since Matlab relies on these codecs, the Matlab *VideoReader* class throws an error when attempting to read a 4K video file. This issue is alleviated by instaling 3rd party codecs.

A second issue that may arise is the use of the *read* function is flagged bt Matlab as subject to removal in future versions of Matlab. This function is used, rather than the new *readFrame* function, as the *read* function enables a frame to be read based on an index value. The *readFrame* function requires you to read each frame sequentially, which takes a lot of unnecessary time when you are reading every N frames.

### 4.1.3 Convert Log file to CSV

The log file that is recorded on the SD card contains L1 GPS data which can be used to provide an initialization for the camera position when running Structure from Motion algorithms. A quick algorithm to decode the log file and convert it from NMEA to a more easily interpretted CSV is provided electronically. An example resultant file is shown in Figure 4.4. This file is simply raw time and GPS position data, but it can be leveraged to interpolate camera positions, as described in 4.1.4.

```
UTCtime(yyyymmddHHMMss.fff),Lat,Lon,fixQuality,nSats,hDOP,AltAboveMSL,geoidHeight
20160304120232.303,44.562283,-123.271350,1,4,1.600,79.400,-21.000
20160304120234.000,44.562200,-123.271383,1,4,1.600,62.100,-21.000
20160304120235.000,44.562150,-123.271417,1,4,1.610,51.000,-21.000
20160304120236.000,44.562100,-123.271450,1,4,1.600,43.900,-21.000
20160304120237.000,44.562050,-123.271433,1,4,1.600,46.600,-21.000
20160304120238.000,44.562000,-123.271433,1,4,1.600,48.600,-21.000
20160304120239.000,44.561967,-123.271417,1,4,1.600,50.200,-21.000
20160304120240.000,44.561933,-123.271417,1,4,1.600,52.000,-21.000
20160304120241.000,44.561917,-123.271400,1,4,1.600,53.600,-21.000
20160304120242.000,44.561917,-123.271383,1,4,1.600,55.500,-21.000
20160304120243.000,44.561900,-123.271383,1,4,1.600,57.500,-21.000
20160304120244.000,44.561883,-123.271383,1,4,1.600,58.800,-21.000
20160304120245.000,44.561850,-123.271383,1,4,1.600,60.300,-21.000
20160304120246.000,44.561817,-123.271367,1,4,1.600,61.300,-21.000
20160304120247.000,44.561767,-123.271350,1,4,1.600,62.600,-21.000
20160304120248.000,44.561717,-123.271350,1,4,2.140,62.500,-21.000
```

Figure 4.4: The log file is turned into a CSV file so that the GPS data can be easily extracted and leveraged.

### 4.1.4 Interpolate GPS data for each image

Once all of the imagery has been extracted, estimated camera pose positions can be calculated using various sources of GPS data. Figure 4.5 demonstrates an example metadata file generated from the logfile gps positions. The algorithm, shown in C.4, is developped to be modular so that any CSV file can be used to add position or other temporal metadata to the imagery. The CSV file must have:

a) headers to describe the data, as these are propagated through to the merged metadata file

b) A first column representing UTC time in the format 'yyyymmddHHMMss.fff.'

If these parameters are met, other sensor data such as that from the autopilot can be easily integrated into the workflow.

```
#Position data interpolated from C:\Users\Richie\Documents\Sandbox\20150831_testreadbinary\Test030416a\LOG00323_gps.txt
ImageName,ActualTime(yyyymmddHHMMss.fff),DesiredTime(yyyymmddHHMMss.fff),Lat,Lon,fixQuality,nSats,hDOP,AltAboveMSL,geoidHeight
00089_GOPR6864.jpg,20160304120755.988,20160304120756.000,44.560100,-123.271283,1.000000,9.000000,0.930000,62.500000,-21.000000
00090_GOPR6864.jpg,20160304120757.992,20160304120758.000,44.560100,-123.271283,1.000000,9.000000,0.930000,62.500000,-21.000000
00091_GOPR6864.jpg,20160304120759.992,20160304120800.000,44.560100,-123.271283,1.000000,9.000000,0.930000,62.500000,-21.000000
00092_GOPR6864.jpg,20160304120801.996,20160304120802.000,44.560100,-123.271283,1.000000,9.000000,0.930000,62.500000,-21.000000
00093_GOPR6864.jpg,20160304120803.996,20160304120804.000,44.560100,-123.271283,1.000000,9.000000,0.930000,62.500000,-21.000000
00094_GOPR6864.jpg,20160304120806.000,20160304120806.000,44.560100,-123.271300,1.000000,9.000000,0.930000,62.500000,-21.000000
00095_GOPR6864.jpg,20160304120808.000,20160304120808.000,44.560100,-123.271300,1.000000,9.000000,0.930000,62.500000,-21.000000
00096_GOPR6864.jpg,20160304120810.004,20160304120810.000,44.560100,-123.271300,1.000000,9.000000,0.930000,62.500000,-21.000000
00097_GOPR6864.jpg,20160304120812.004,20160304120812.000,44.560100,-123.271300,1.000000,9.000000,0.930000,62.500000,-21.000000
00098_GOPR6864.jpg,20160304120814.008,20160304120814.000,44.560100,-123.271300,1.000000,9.000000,0.930000,62.500000,-21.000000
00099_GOPR6864.jpg,20160304120816.008,20160304120816.000,44.560083,-123.271300,1.000000,9.000000,0.930000,62.500000,-21.000000
00100_GOPR6864.jpg,20160304120818.012,20160304120818.000,44.560083,-123.271283,1.000000,9.000000,0.930000,62.500000,-21.000000
```

Figure 4.5: The GPS data is appended to the image position metadata so that each image has interpolated GPS values.

## 4.2 Temporal Accuracy

The temporal accuracy was tested by observing a cell phone that was updating with the UTC time via a website. This test is limited greatly by a number of factors, included refresh rate of the phone screen, refresh

rate of the website, and the displayed temporal resolution of the website is only to the second. Initial results show that the time is correctly synchronized to UTC time, in the sense that the times appear to match relatively well with what is shown on the screen, as shown in Figure 4.6.

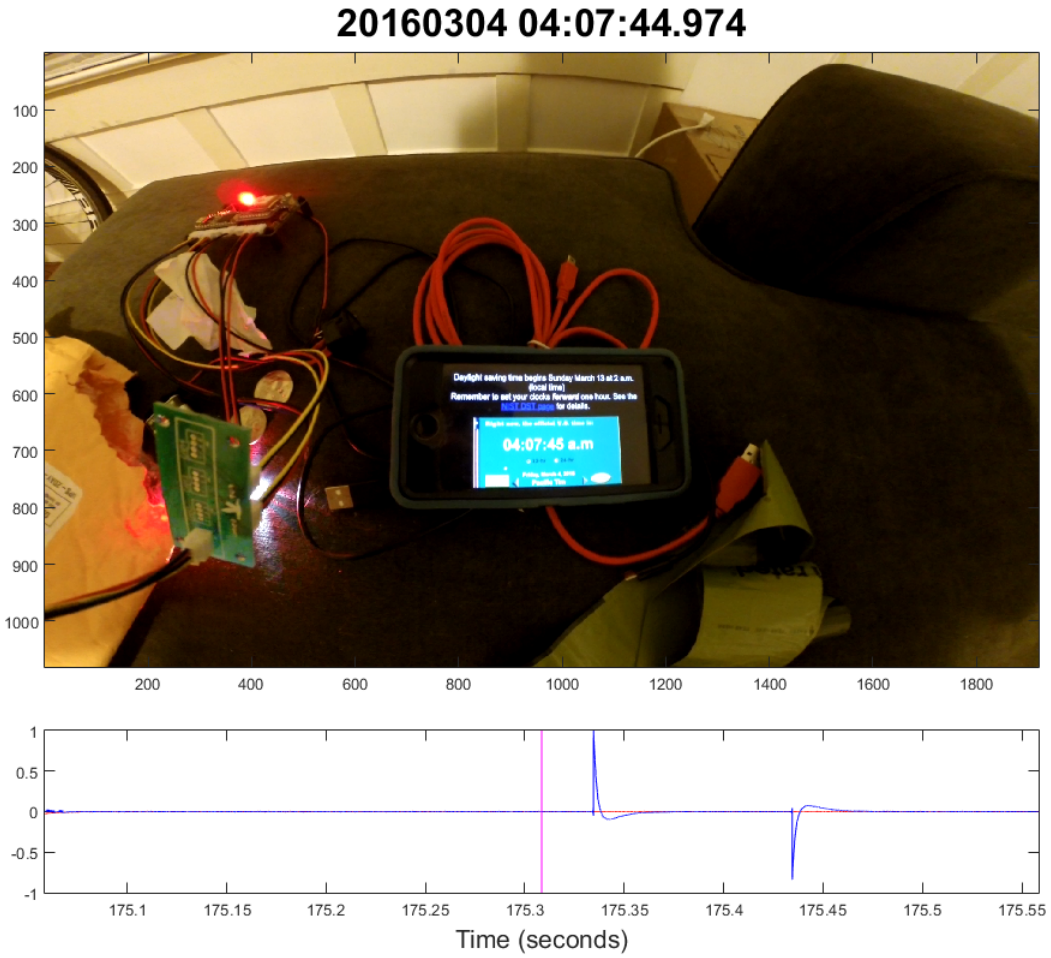



Figure 4.6: The synchronization of the imagery was tested using a cell phone updating via a time server to UTC time. The top pannel shows the imagery, and the bottom panel shows the PPS signal channel of the audio. The magenta line represents the current time. Notice that the white, PPS light under the green PCB is on, before the signal is shown in the audio.

However, one issue that was observed is the latency between the video and the audio channel within the GoPro. GoPro cameras are consumer grade cameras, and therefore time synchronization between the audio and video does not need to be perfectly precise. Notice in Figure 4.6 how the white PPS light under the PCB is lit up before it is recorded on the audio channel. This indicates that the Video signal is farther ahead in time than the audio channel. This offset was observed to be consistently one to two frames off, but further tests would need to be done to make any statements about it's consistency. Hopefully, the latency between the audio and video port is a constant offset that can be calculated once and applied to each camera. Further tests will need to be done to fully quantify the accuracy of this methodology due to other sources of error.

## 4.3 Possible Improvements

### 4.3.1 Embed x8 Flight log info

The X8 Pixhawk autopilot records position and orientation values directly to the SD card in the Pixhawk. These data can be read into the 3D Robotics "Mission Planner" software, and converted into a Matlab file. Once the data is in Matlab, it should be saved as a CSV, as described in 4.1.3. While the autopilot is not rigidly mounted to the cameras, it should provide a more accurate camera pose estimation than what could be derived from the onboard 9dof IMU.

### 4.3.2 Incorporate onboard IMU

If however, the vibration damping between the camera and the X8 autopilot significantly invalidates the "rigid body" assumption, the onboard IMU could be used to generate improved position and orientation values. A Kalman filter should be implemented to generate an improved trajectory than what could be generated from just the raw data. One issue that will need to be overcome in order to integrate this system, is to modify the Teensy code to record the raw 9dof measurements. These raw values have been excluded from this iteration, as there was concern that reading and writing from another sensor could cause latency and delays in the time synchronization section of the code. Further tests will be required to ensure that this is not the case.

# Chapter 5

# Acquisition and Data Storage

## 5.1 Mission Planning Tool

To aid in development of a mission plan, a flight planning tool was developed to assess dwell time at various altitude and speeds. The tool, *mvss_cameraFlightPlanning2*, allows the user to manually input flight parameter constants of altitude, flight speed, and image frame rate, as shown in Algorithm 5.1.

Algorithm 5.1: The flight planning tool is used to assist with calculating dwell time and spatial coverage for various flight parameters.

```
function mvss_cameraFlightPlanning2()
%% Use this script to generate some plots to aid in your flight planning
%% Flight Parameter Constants
altmeters=200;%altitude in meters
FlightSpeed=10;%flight speed in meters/second
imagedt=0.5;%difference in seconds between images
maximumCaxis = 300; %time in seconds
```

The *mvss_cameraFlightPlanning2* algorithm automatically generates two plots, shown in Figure 5.1. The left plot is a spatial snapshot, showing what one image from each of the four cameras will cover spatially when it is centered at (0,0). In the example shown, the images cover the entire 4km grid in the along-track distance. The right plot shows the time dwell that each location would be imaged assuming a constant flight speed and altitude. Higher, red values, represent a longer dwell time. Each camera configuration yields a slightly different heat map of values, which can be useful for planning flight speeds and planning coverage.

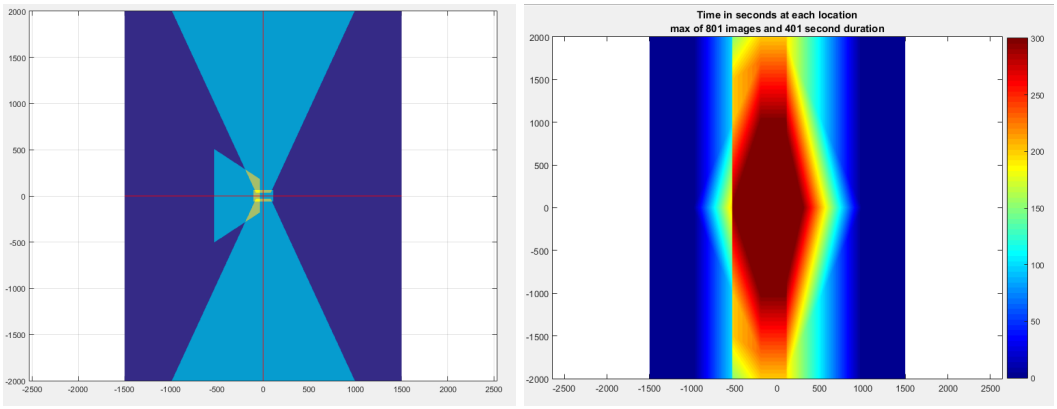



Figure 5.1: (Left) The spatial extent of one image snapshot is plotted with cross-track distance on the X-axis, and along-track distance on the Y-axis. (Right) The dwell time at every location in the cross-track(x-axis) and along-track(y-axis) is colored based on time in seconds, assuming a constant flight speed and altitude.

## 5.2 Flight Checklists

Three initial checklists are provided to ensure accurate and reliable data acquisitions. The first checklist represents what should be done prior to a flight. This includes checking all connections, batteries, SD cards, and settings to find any bugs before the actual acquisition. The second checklist is to be used during the actual data collect. This list should be improved to include more detailed X8 flight checklists, provided by 3DR robotics. The final checklist describes how data should be stored and organized after a data collect. This organization of data will make processing much more intuitive and user friendly. These checklists should be seen as an initial start, and should be iterated upon and improved based on user experience.

### 5.2.1 Pre Flight Checklist

**Day Before Flight**

- ☐ Charge GoPro Batteries
- ☐ Check Voltage/Charge PCB Batteries
- ☐ Clear SD Card for each camera
- ☐ Insert SD card for each camera
- ☐ Test Electronics to ensure it turns on and acquires PPS
- ☐ Clear SD card for electronics (DO NOT CLEAR CONFIG FILE)
- ☐ Make sure the config file still says 115200 Baud
- ☐ Place SD card for electronics into PCB slot
- ☐ Install Sensor on X8
- ☐ Charge Battery for X8
- ☐ Upload Mission to X8

**Once batteries are charged**

- ☐ Install electronics tray
- ☐ Turn on electronics and ensure power light comes on and voltage is good
- ☐ Turn off electronics so power light goes off
- ☐ Ensure cameras turn on and are functional
- ☐ Ensure camera settings are set to 4K mode
- ☐ Place cameras in enclosures
  - ☐ Make sure each camera is in it's corresponding spot based on focal length
- ☐ Lock down each camera with screw
- ☐ Ensure cameras turn on and are functional after locking down
- ☐ Turn cameras all off so not to drain battery
- ☐ Place Cap on cameras to protect lenses
- ☐ Plug in the trigger cables, and handle cable management
- ☐ Plug in external GPS, and handle cable management
- ☐ lock down electronics tray with screw

**One Hour before Flight**

- ☐ Check Electronics
  - ☐ Turn on electronics and ensure power light comes on and voltage is good
  - ☐ Turn off electronics so power light goes off
- ☐ Check Cameras
  - ☐ Ensure cameras turn on and are functional
  - ☐ Ensure camera settings are set to 4K mode
  - ☐ Ensure SD cards are in cameras
  - ☐ Power off all cameras
- ☐ Check X8
  - ☐ Check X8 Battery Voltage

### 5.2.2 Flight Checklist

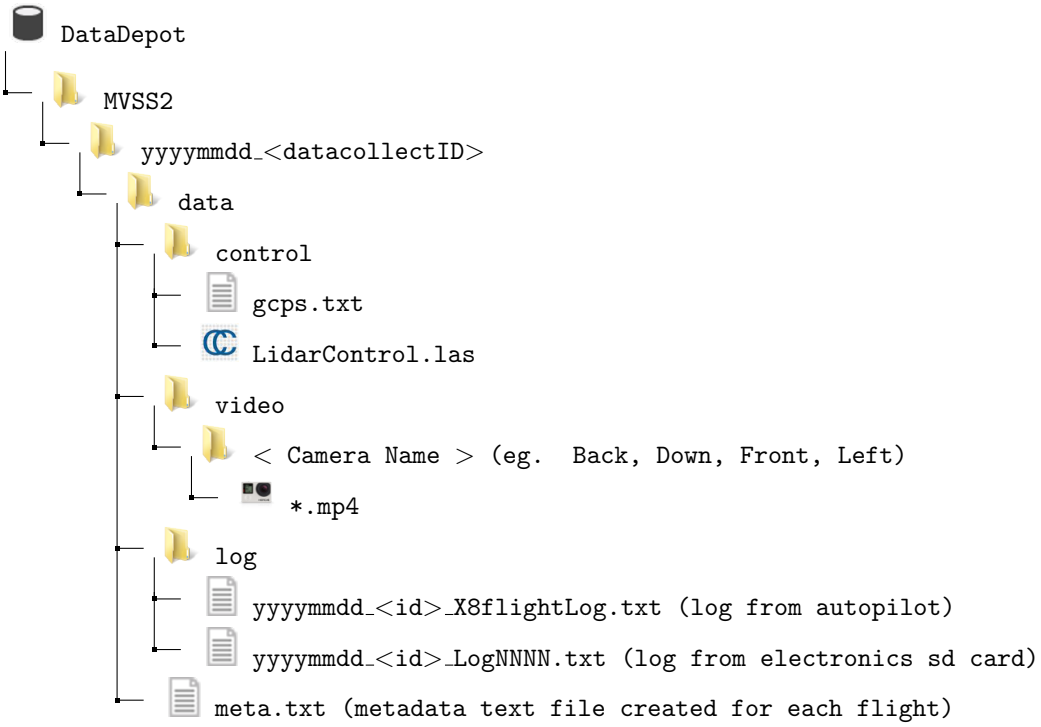
- ☐ Place X8 with sensor where you want to take off from
- ☐ Unscrew lens caps
- ☐ Power on Electronics. Do not power off until done with collect.
- ☐ Wait until White Light is Flashing, signifying PPS (may take a minute)
- ☐ Power on X8 and initialize
- ☐ Wait until X8 is initialized and ready to takeoff
- ☐ Turn on Each camera
- ☐ Ensure each is set to 4k mode
- ☐ Press record on each camera
- ☐ Set X8 flying
- ☐ Land X8
- ☐ Stop recording on each camera
  - ☐ Make note of any cameras are off
  - ☐ Make note of the PPS status
- ☐ Turn off the Electronics via the panel switch
- ☐ Turn Off X8
- ☐ Place Lens Caps on sensor to protect lenses

## 5.3 Post Flight Checklist

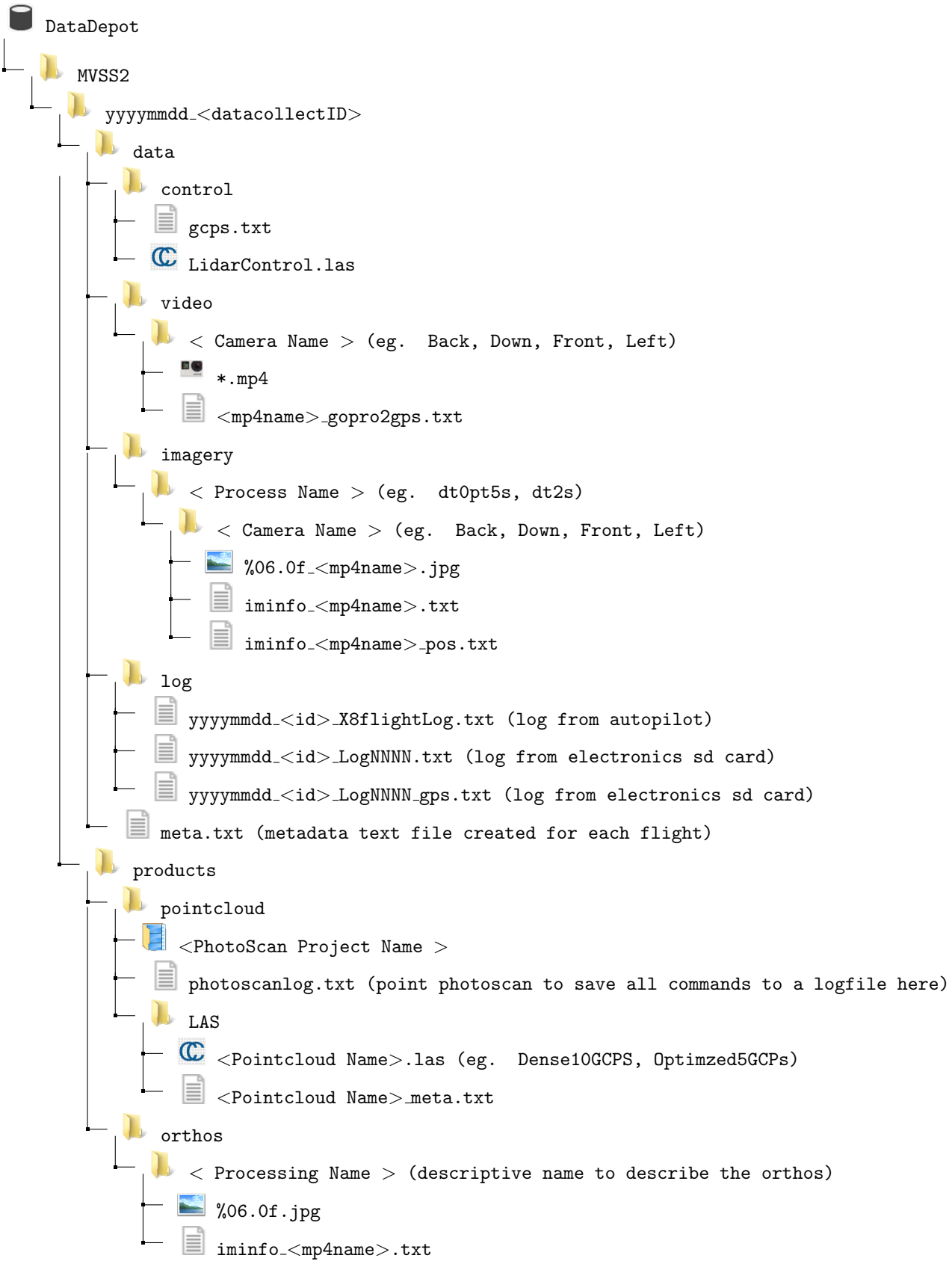
- ☐ Extract Data and organize, per 5.4.1
  - ☐ Different 'data collect ID' should be saved for each collect on a day
  - ☐ Extract the logfileNNN.txt from PCB SD card
  - ☐ Extract Each Camera *.MP4 files
  - ☐ Extract SD card, from the X8FlightLog from the autopilot
- ☐ Task someone with processing the Lidar or GPS surveys
- ☐ Charge GoPro Batteries
- ☐ Charge PCB battery
- ☐ Double check all data is backed up
- ☐ Triple check all data is backed up
- ☐ Clear SD cards
  - ☐ Clear Camera SD cards
  - ☐ Clear Log SD card (except config file)
  - ☐ Clear X8 SD card
- ☐ Place SD cards back in cameras, PCB, and X8
- ☐ Store equipment

## 5.4 Data Management

### 5.4.1 Raw Data Structure



### 5.4.2 Processed Data Structure

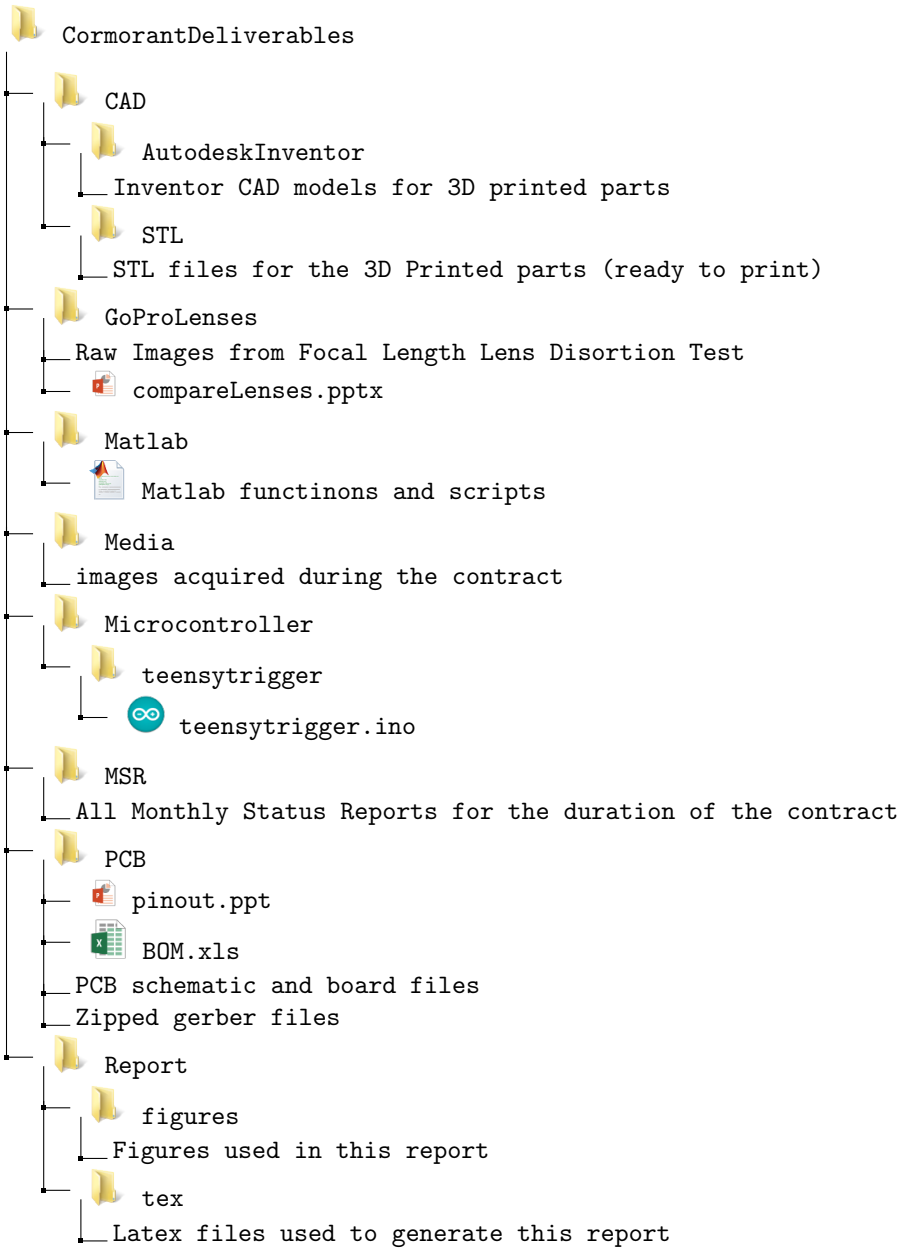


# Acknowledgement

Cormorant Analytics, Inc. expresses appreciation to the FRF for funding support to this contracted effort and for the opportunity to contribute to the advancement of multi-view stereo generated point clouds for increased situational awareness. Cormorant Analytics, Inc. also expresses appreciation to Integrity Applications Incorporated for their support and assistance throughout the effort.

# Appendix A

# Files Delivered Electronically



# Appendix B

# How to Order a PCB Designed using EAGLE

PCBs are designed in various programs, and gerber files can be seen as a pdf type of file. Gerber files are a number of generated files that are universally accepted by board manufacturers. These files tell the board manufacturer exactly how to cut, drill, silkscreen, and route traces on the PCB. These gerber files have been generated and provided, so to order another PCB the following steps must be taken:

1. Navigate to `https://www.apcircuits.com/ap/webclient`
2. Log in and fill out the shipping information
3. Fill out the board information as shown in
4. Upload a zip file of the gerber files and select the layer correspondences as shown

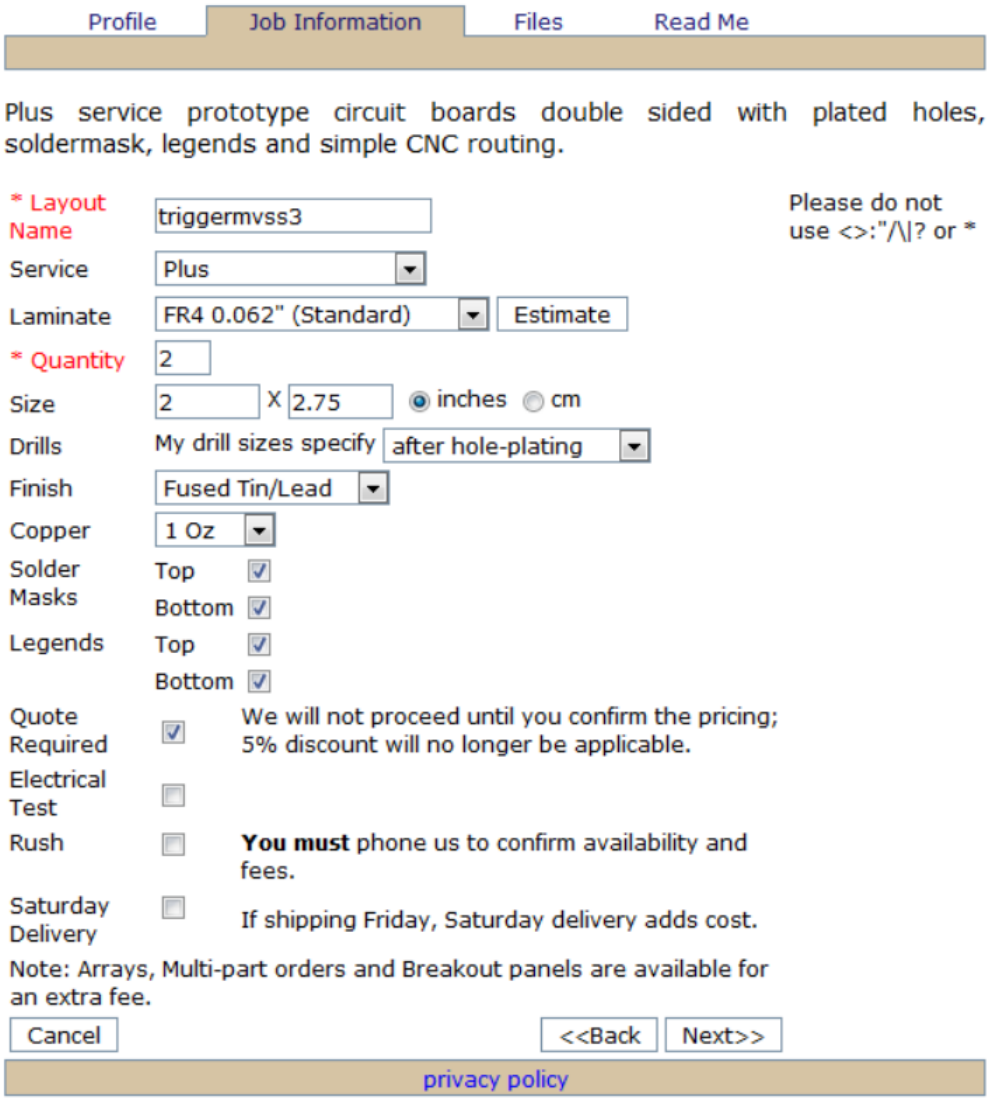



Figure B.1: Fill out the information as shown to generate a two sided board with silkscreen on both sides.

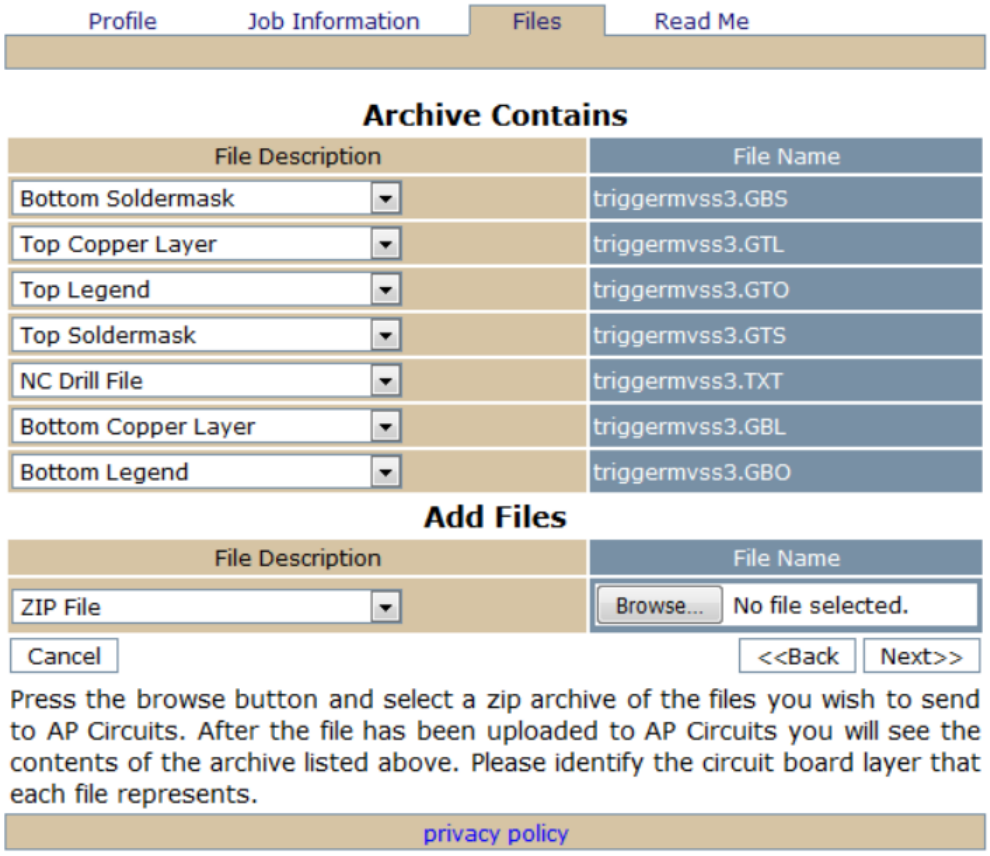



Figure B.2: Select the correspondences based on the file extension so that the board is manufactured correctly.

# Appendix C

# Algorithms

## C.1 function getValue

Algorithm C.1: The function getValue is used to parse the NMEA data to extract relevant data.

```
function X=getValue(data,id,columnNum,isstring,alsocontains)
%% Retreive data from an ascii vector where each line starts with a '$<id>'
%
%  Input:
%   data: 1xM: vector of characters read with fileread(filename) command
%   id: 1xN: vector of characters that identify the id
%   columnNum: 1 {int}: index of the column number of data to retreive
%   isstring: boolean: return value as a string or double?
%   alsocontains: 1xP : vector of characters that also must be in each line
%
%  Output:
%   X: 1xQ : vector of all output data found, eith as string or double
%
%  Example:
%   data = fileread('test.txt');
%   X = getValue(data,'$MSG',5,1,'$GPGGA');

%% Handle Inputs
columnNum=columnNum+1;
numchars=length(id);
if nargin==3
    isstring=0;
    alsocontains=0;
end
if nargin==4
    alsocontains=0;
end
%% Find End of lines, and determine if that line met the search criteria
ind=[0 strfind(data,char(10))];
NumX=0; %count number of X values found
for i=1:length(ind)-1
    linestring=data(ind(i)+1:ind(i+1)-2);
    if strcmp(linestring(1:numchars),id) && ...
            (~isempty(strfind(linestring,alsocontains)) || ...
            sum(alsocontains)==0)
```

```
36          NumX=NumX+1;
37      end
38  end
39  %% Preallocate
40  if isstring
41      X=cell(1,NumX);
42  else
43      X=nan(1,NumX);
44  end
45  count=0;
46  %% Search through each line and populate data if it matches the id
47  for i=1:length(ind)-1
48      linestring=data(ind(i)+1:ind(i+1)-2);
49      if strcmp(linestring(1:numchars),id) && (~isempty(strfind(linestring,
            alsocontains)) || sum(alsocontains)==0)
50          count=count+1;
51          AllValues=textscan(linestring,'%s','delimiter',' ,');
52          if length(AllValues{1})<columnNum
53              error(['ID: ' id ' doesnt have a column ' num2str(columnNum)]);
54          end
55          if isstring
56              X{count}=(AllValues{1}(columnNum));
57          else
58              X(count)=str2double(AllValues{1}(columnNum));
59          end
60      end
61  end
62
63  end
```

## C.2 function calcGopro2GPStime

Algorithm C.2: The function getValue is used to parse the NMEA data to extract relevant data.

```
function calcGopro2GPStime(mp4name,logfilename,dodebug)
%% Calculate the linear fit to go from gopro time in seconds to GPS datenum
% time in days.  Save the output as a text file based on the mp4name and
% saved location
%
%  Input:
%   mp4name: string : absolute file location of the mp4 file
%   logfilename: string : absolute file path of the log file
%   dodebug: logical {0 1} : If true, debug plots will appear
%
%  Saved File:
%   <mp4name>_gopro2gps.txt
%      - contains P1 and P2, the slope and y intercept of a linear fit
%      - also contains metadata about the fit
%% Set constants
BAUDRATE=1660; %baud rate is set to 300, but actually is 1660
NMEADT = 1; % time between signal pulses
ALGORITHMVERSION = 'V0.1';
%% Read audio from mp4name
[y,Fs]=audioread(mp4name);
pps=y(:,2);
nmea=y(:,1);
% make time variable
t=0:1/Fs:(numel(pps)-1)*1/Fs;

%% Calculate goPro Index Count Values and times
[goproInd.tGP,goproInd.val] = ...
    calcGoproNmeaCounts(t,nmea,NMEADT,BAUDRATE,dodebug);
%% Calculate goPro PPS rising and falling edge times
[goproPPS.tGPRising,goproPPS.tGPFalling]=calcPPStimes(t,pps,dodebug);

%% Read SD Card Data with GPS times, PPS times, and Count Index Times
[logGPS,logPPS,logInd]=getLogData(logfilename);

%% Calculate Offset from GoPro time to GPS time
[GoPro2utcConstants,npps] = ...
    calcGoProGPS(logInd,logGPS,logPPS,goproInd,goproPPS,dodebug);

%% Save Output to text file
[d,f,~] = fileparts(mp4name);
outname = [d '/' f '_gopro2gps.txt'];

fid = fopen(outname,'w+t');
fprintf(fid,'P(1),%.20f\n',GoPro2utcConstants(1));
fprintf(fid,'P(2),%.20f\n',GoPro2utcConstants(2));
fprintf(fid,'nPPS,%.0f\n',npps);
fprintf(fid,'Version, %s\n',ALGORITHMVERSION);
%calculate minimum and maximum times
minT = polyval(GoPro2utcConstants,0);
maxT = polyval(GoPro2utcConstants,max(t));
```

```
51 fprintf(fid,'MinTime, %s\n',datestr(minT));
52 fprintf(fid,'MaxTime, %s\n',datestr(maxT));
53 fprintf(fid,'Duration, %.2f\n',max(t));
54 fclose(fid);
55
56 end
```

## C.3 function extractFrames

Algorithm C.3: The function getValue is used to parse the NMEA data to extract relevant data.

```
function extractFrames(mp4name,outfolder,extractTimesGPS,fitfilename,DODEBUG)
%% Extracts Images from the mp4 file at the extractTimeGPS times and saves
% them to an outfolder.  Uses fitfilename to convert from gopro time to GPS
% time based on a linear fit.
%
%  Input:
%   mp4name: string : filename of mp4 gopro video
%   outfolder: string : folder to save imagery to
%   extractTimesGPS: 1xN : vector of time in GPS to extract frames at
%   fitfilename: string : absolute filename of the linear fit data
%
%  Saved Files:
%   *%010.0f_<outfolder>.jpg : saves N images, less if time is out of range
%   imageinfo.txt: CSV + header with imname, extractTimesGPS, actualTimes

%% Set Constants
MAXDTCONST = 1.05; %percentage above nyquist allowable
nFrames = numel(extractTimesGPS);
%% Read mp4 file
mov = VideoReader(mp4name);
[~,justmp4name,~]=fileparts(mp4name);
%% Read Fit filename
data = fileread(fitfilename);
dataParts = strsplit(data,{',','\n'});
gopro2gps = str2double([dataParts(2); dataParts(4)]);
%% Convert mp4 times to GPS times
tGoPro = 0 : 1/mov.FrameRate : mov.Duration;
tGPS = polyval(gopro2gps,tGoPro);
frameinds = 1:numel(tGPS);
%% Calculate indices to extract
MAXDT = (1/mov.FrameRate/2)*MAXDTCONST;
frame2extract = interp1(tGPS,frameinds,extractTimesGPS,'nearest');
isBad = isnan(frame2extract);
frame2extract(isBad)=1;
residuals = (tGPS(frame2extract)-extractTimesGPS)*60*60*24;
isOutOfRange = abs(residuals)>MAXDT | isBad;

if DODEBUG
    figure
    plot(residuals);
    hold on
    plot(residuals,'g*');
    iRes = 1:numel(residuals);
    plot(iRes(isOutOfRange),residuals(isOutOfRange),'r*')
    ylim([-MAXDT*2 MAXDT*2])
    xlim([1 numel(extractTimesGPS)])
    title('Actual Frame Time - Desired Frame Time');
    ylabel('Time Difference (seconds)');
    xlabel('index');
end
```

```
%% Make directory to save images to
mkdir(outfolder);
%% Make text file to save images to
outfilename = [outfolder '/iminfo_' justmp4name '.txt'];
fid = fopen(outfilename,'w+t');
fprintf(fid,'ImageName,ActualTime(yyyymmddHHMMss.fff)');
fprintf(fid,',DesiredTime(yyyymmddHHMMss.fff)\n');
%% Loop over each index
if DODEBUG
    f = figure;
end
for iTimestep = 1:nFrames
    if ~isOutOfRange(iTimestep)
        iFrame = frame2extract(iTimestep);
        % Extract indices and save Images based on index number
        I = read(mov,iFrame);
        %   ^says it will be removed, but readframe cant extract at exact
            index
        imname = sprintf('%s/%05.0f_%s.jpg',outfolder,iTimestep,justmp4name);
        imwrite(I,imname);
        % write image info to text file
        [~,imjustname,ext]=fileparts(imname);
        fprintf(fid,'%s,%s,%s\n',...
            [imjustname,ext],...
            datestr(tGPS(iFrame),'yyyymmddHHMMss.fff'),...
            datestr(extractTimesGPS(iTimestep),'yyyymmddHHMMss.fff'));
        fprintf('%s\n',imname);
        fprintf('%s,%s,%s\n',...
            [imjustname,ext],...
            datestr(tGPS(iFrame),'yyyymmddHHMMss.fff'),...
            datestr(extractTimesGPS(iTimestep),'yyyymmddHHMMss.fff'));
        if DODEBUG
            figure(f)
            image(I);
            title({datestr(tGPS(iFrame),'yyyymmdd    HH MM:ss.fff'),...
                datestr(extractTimesGPS(iTimestep),'yyyymmdd    HH MM:ss.fff')})
                ;
            drawnow
        end
    else
        fprintf('%05.0f: %s Out Of Range\n',iTimestep,datestr(extractTimesGPS(
            iTimestep)));
    end
end
%% close iminfo file
fclose(fid);
end
```

## C.4 function addPositionInfo

Algorithm C.4: The function getValue is used to parse the NMEA data to extract relevant data.

```
function addPositionInfo(imageinfo,positiondata)
%% add gps-ins position data the imageinfo text file based on times
%
%  Input:
%   imageinfo: string :absolute filename for which to add position info to
%   gpsdata: string : absolute filename for a file with time/position info
%
%  Saved File:
%   imageinfo_2:
%    - modifies imageinfo and resaves it with more data as _2

%% Read Imageinfo and extract times
iminfodata = importdata(imageinfo);
imnames = iminfodata.textdata(2:end,1);%first line is header
imtimesDatenum = nan(1,numel(imnames));
for i=1:numel(imnames)
    imtimesStr = sprintf('%.3f\n',(iminfodata.data(i,2)));
    imtimesDatenum(i) = datenum(imtimesStr,'yyyymmddHHMMss.fff');
end
nImages = numel(imnames);
%% Read positiondata and extract times, header, other variables
xyzdat = importdata(positiondata);
positionTimeNum = xyzdat.data(:,1);
positiontimesDatenum = nan(1,numel(positionTimeNum));
for i=1:numel(positionTimeNum)
    strTime = sprintf('%.3f\n',positionTimeNum(i));
    positiontimesDatenum(i) = datenum(strTime,'yyyymmddHHMMss.fff');
end
allposdat = xyzdat.data(:,2:end);
allposHeaders = xyzdat.colheaders(2:end);
nPositionColumns = size(allposdat,2);
%% Interpolate position data onto imageinfo times
InterpPos = nan(nImages,nPositionColumns);
for iColumn=1:nPositionColumns
    InterpPos(:,iColumn) = ...
        interp1(positiontimesDatenum,allposdat(:,iColumn),imtimesDatenum);
end
%% rewrite and save imageinfo with new info, save as new file _2
[dname, fname, ~] = fileparts(imageinfo);
outname = [dname '\' fname '_Pos.txt'];
fid = fopen(outname,'w+t');
%print header
fprintf(fid,'#Position data interpolated from %s\n',positiondata);
fprintf(fid,'ImageName,ActualTime(yyyymmddHHMMss.fff),');
fprintf(fid,'DesiredTime(yyyymmddHHMMss.fff)');
for i=1:nPositionColumns
   fprintf(fid,',%s', allposHeaders{i});
end
fprintf(fid,'\n');
%for each image, print a line with interpolated info
```

```
for iIm=1:nImages
    fprintf(fid,'%s,%.3f,%.3f',imnames{iIm},iminfodata.data(iIm,1),...
        iminfodata.data(iIm,2));
        fprintf(fid,',%.6f',InterpPos(iIm,:));
        fprintf(fid,'\n');
end
fclose(fid);
end
```