LOW-COST PHOTOGRAMMETRY SYSTEM FOR GEOREFERENCED STRUCTURE FROM MOTION

By

Nicholas Hill

July 2022

Director of Thesis: Dr. Zhen Zhu

Department of Engineering

ABSTRACT

Photogrammetry has become increasingly available as a tool for re-constructing the 3D model of real-world objects and terrain, which is referred to as the structure from motion (SFM) solution. The re-constructed model has three-dimensional colorized surfaces. Integration of geo-referenced data into SFM allows for precise geo-registration, scaling and orienting of the object. Today, industrial users of SFM typically acquire imagery from unmanned aerial vehicles (UAVs). UAV-based SFM relies on having a large number of ground control points that were surveyed ahead of time as the main source of geo-referenced data. The deployment and survey of ground control points are time consuming, and sometimes infeasible due to environmental constraints. A better solution is to gather location and/or orientation data of the camera in flight. A capable navigation device can record the precise location and orientation of the camera at the exact moment at which every image was taken. With that information, few or no ground control points are needed for geo-registration. However, the commercially available solutions of UAV-based SFM with an on-board navigator tend to be bulky and expensive. A low-cost, compact solution with open interfaces will be proposed in this work.

LOW-COST PHOTOGRAMMETRY SYSTEM FOR GEOREFERENCED STRUCTURE FROM MOTION

A Thesis

Presented to the Faculty of the Department of Engineering

East Carolina University

In Partial Fulfillment of the Requirements for the Degree

Master's of Science in Mechanical Engineering

by

Nicholas Hill

July, 2022

LOW-COST PHOTOGRAMMETRY SYSTEM FOR GEOREFERENCED STRUCTURE FROM MOTION

By

Nicholas Hill

APPROVED BY:

Director of Thesis	Zhen Zhu, PhD
Committee Member	Teresa Ryan, PhD
Committee Member	Rui Wu, PhD
Committee Member	Name and Degree of Committee Member
Committee Member	Name and Degree of Committee Member
Chair of the Department of Engineering	Barbara Muller-Borer, PhD
Dean of the Graduate School	Interim Dean Dr. Kathleen Cox

DEDICATION

This work is dedicated to my wife and son. I could not have completed this work without their patience and understanding.

ACKNOWLEDGMENT

I would like to thank the NC Department of Transportation and the US Airforce for partially funding my thesis work. I would also like to thank Dr. Zhen Zhu for his guidance and support in the creation of this work. Finally, I would like to thank my committee members, Dr. Teresa Ryan and Dr. Rui Wu for their input and contributions to this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS/ABBREVIATIONS

3D	Three Dimensional	1
DEM	Digital Elevation Modeling	56
FDM	Fused Deposition Modeling	43
FPS	Frames Per Second	19
GCP	Ground Control Point	2
GNSS	Global Navigation Satellite System	2
GPIO	General Purpose Input Output	31
IDE	Integrated Development Environment	27
IMU	Inertial Measurement Unit	3
LiDAR	Light Detection and Ranging	48
MVE	Multi View Environment	3
PETG)	Polyethylene Terephthalate Glycol	43
PPK	Post Processed Kinematics	5
PPS	Pulse Per Second	31
RTK	Real Time Kinematics	2
SFM	Structure From Motion	1
UAV	Unmanned Ariel Vehicle	2

Chapter 1 INTRODUCTION

Photogrammetry refers to the technology of measuring real-world objects via imagery [1]. Photogrammetry can be used for two-dimensional or three-dimensional estimation of topographical features, point cloud recreation of objects, or top-down maps of large areas. Different methods exist to digitally recreate these features.

In this work, the focus is on the photogrammetry methods that can be used to integrate multiple images together to form a three-dimensional (3D) model. In these methods, multiple images are typically related to each other by using point features that are common to these images. Furthermore, if the points in a known 3D model can be geo-referenced, this process is referred to as geo-registration in photogrammetry [2]. There are multiple algorithms available for geo-registration. The optimum choice of a target application depends on the number of cameras available, referencing method, number of images, and calculation time allowed [3].

Today, structure from motion (SFM) is the most commonly used technique for reconstructing 3D models in photogrammetry. SFM makes use of a large number of images of the same scene or objects, taken at different angles and distances. The images are geometrically related to each other by correlating common point features. Based on the observations of these common point features in the images, an estimator can be used to compute the camera pose (location and orientation) for each image, as well as the 3D location of the point features [4]. During this process, the camera is expected to move around a fixed object or scene, and the 3D model of the object or the scene can be reconstructed as a point cloud, which includes a large number of 3D point features.

SFM is widely used by research communities, various industries and government users. It only uses a single camera, as opposed to a stereo-vision system that requires two cameras. To properly acquire the images needed for SFM over a large area, it requires the ability to quickly move the camera and take pictures at various angles and distances. Therefore, a camera system based on unmanned ariel vehicle (UAV) has become the dominant choice for SFM by many users. Modern small UAVs are easy to control and maneuver and can carry high-quality onboard cameras. The UAV camera systems are considered low-cost in a lot of industrial settings [5]. Although there have been many regulations regarding the safe operations and airspace restrictions of hospitals, airports, and military bases, small UAV can still be operated to meet the needs of most users [6].

However, the 3D model and the camera pose estimated in SFM are all relative, since they are referenced in a scaleless coordinate system defined relative to the camera itself. Therefore, the 3D model cannot be directly placed in a real-world coordinate system.

Pre-surveyed ground control points (GCPs) or precise pose information of the camera can be used to provide the link between SFM and a world coordinate [5]. If part of the point features used in SFM are captured from known objects at pre-surveyed locations in a world coordinate system, they will be identified as geo-referenced markers. These markers can be used to estimate the location of all the 3D points from SFM in the same world coordinate system.

On the other hand, the precise pose the camera recorded in a world coordinate system can be used to estimate a geo-registered 3D model as well. Precise camera location can be achieved using the global navigation satellite systems (GNSS), which can achieve centimeter-level accuracy using real time kinematics (RTK) technologies. While generally considered expensive, newer technologies are reducing the cost of RTK systems [5]. Orientation of a camera can be sensed using an inertial measurement unit (IMU) that is typically integrated with a satellite navigation system on a small UAV. Precise camera position and orientation are crucial to proper scaling, orientation and geo-registration of the 3D model in the world frame. However, even if the orientation measurement is not available to small UAVs, geo-registration can still benefit from precise positioning alone.

There are multiple software options that automatically perform SFM with GCPs and/or camera pose, including commercial and open-source libraries. Some of the more common commercial solutions include Agisoft Metashape, use-specific software from PIX4D [7], and 3DF Zephyr [8]. MultiView Environment, (MVE), MicMac, and OpenMVG are all free software capable of SFM [9].

The GNSS receivers on small UAVs are typically used for navigation and control. The accuracy of these GNSS receivers is not always sufficient for SFM. If they were to be used in SFM, the position errors will directly impact the accuracy of geo-registration. A GNSS RTK solution offers centimeter-accuracy, which is ideal for geo-registration in most applications. However, a complete UAV-based geo-registration system would require tight coupling and precise synchronization between the RTK system and the camera system. As a result, there exists few complete geo-registration solutions that make use of precise camera pose in the UAV-photogrammetry industry today. The available systems typically use closed proprietary data interfaces and cannot interchange data with other systems. These systems can be very expensive.

However, much of the work developed by the photogrammetry research community has become open-source or free. They can be used to integrate SFM with precise camera pose. In addition, low-cost GNSS RTK systems have also become available [10].

Therefore, it is now feasible to construct a low-cost standalone SFM system, which includes a commercial off-the-shelf GNSS receiver with a low-cost yet high-resolution camera. It could be attached to any small UAV that is capable of carrying the payload, and the data can be processed in free SFM software. There will be open data interfaces between the GNSS RTK, camera and the SFM software. It will not be dependent on specific manufacturer for each of the components, including the UAV. Instead, users can customize the system based on specific needs. For example, most of the SFM solutions are computed in post-processing software. As a result, the real-time GNSS solution is not always necessary. A post-processed RTK solution can be more accurate and can be better integrated with SFM. It does not require a live data link during operation, which makes it more robust. The design and implementation of an affordable, standalone geo-registration system that uses independent UAV, camera, RTK and SFM modules will be investigated in the work.

An open-source solution for SFM modeling could be used as a cost-effective research tool for academic, government, or personal recreations of three-dimensional features. Prior to hardware selection, some key questions must be answered about the performance capabilities required. In an open-source, open-interface solution where the components are not tightly coupled, it may be more challenging to achieve the expected accuracy in SFM. The performance requirement for each of the components will be defined based on the expected accuracy and the open nature of this solution.

Prior to selecting a cost-effective camera, the capture frame rate must be sufficient to properly capture images and tie those images as closely as possible to the pose data received by the GNSS real time kinematic solution. It is because a low-cost camera that is not tightly coupled with the GNSS device may need to be synchronized to GNSS via a high frame rate, instead of an internal triggering mechanism. However, faster frame rates in a camera generally indicate an increase in price. What is the frame rate required to properly capture images while maintaining the minimization of cost requirement?

After selecting the camera, it must then be able to accurately coordinate images to RTK GNSS data. The image captured must be as close as possible to the instant the positional data is valid in order to minimize propagating errors in the model. How can the RTK GNSS positional data be properly tied to the correct image in a standalone system and properly recorded for post processing?

The geo-registration system requires capturing and recording of positional data received from GNSS. While a real time solution is not necessary, the data must be recorded rapidly as it occurs in order to ensure data integrity for later processing. Data from a ground reference receiver must be used during post processing for the post processed kinematic (PPK) solution. While similar to the system used in the onboard photogrammetry module, it must be free standing and capture and record positional data as well. How can the positional data be captured and recorded rapidly for post processing into an PPK solution?

A standalone system must be able to supply power to itself, and not rely on any outside source. If attached to an UAV, battery weight will also be a design concern. The system may require different voltages for the various systems as well. What voltages are required and how much storage capacity is required for a sufficient recording time to capture the necessary datapoints? How will this be routed to the various systems? Can the UAV properly support the added weight while in flight?

SFM has multiple well-established software solutions. The ideal SFM software will have the ability to incorporate GNSS/RTK data, the lever arm between camera and the GNSS antenna, GNSS coordinates of ground control points. In addition, it should be able to calibrate different camera lenses on the fly. What software options can satisfy the outlined requirements?

In order to assess the quality of the SFM solution, ground objects with pre-surveyed coordinates will be used. How to design and deploy these objects, such that the SFM precision and accuracy can be correctly assessed? How is performance of this system compared to other geo-registration solutions, such as GCP-based SFM or laser-based 3D imaging?

A standalone open-interface photogrammetry module improves over current market offerings in several important ways. A photogrammetry module that is not tied to any specific brand of UAV can be attached to any airframe that is capable of carrying an additional payload within the range of the module. An open-source, open-interface design bears significantly lower cost compared to the complete solution offered by main UAV manufacturers (such as DJI). Finally, components can be changed, modified and upgraded with the open-interface system, as SFM technology advances.

Compared against traditional ground-based survey techniques, UAV-based photogrammetry has been identified as a much more efficient method. Furthermore, since this system reduces the dependency on GCPs, it provides a unique advantage in scenarios where on GNSS challenged locations, or for locations where GCP’s cannot physically be placed in the area of interest. The PPK solution provides higher accuracy in three-dimensional position than standalone GNSS or RTK. It does not rely on a live data link like RTK and other real-time GNSS systems. It is less expensive and more reliable.

Since this system has open interfaces and relies only on commercially available components, there will be additional data format conversion steps needed. Data from GNSS receivers and camera will be converted into standard format, and the relative timing between both data sources has to be precisely represented in the converted data. The converted data are post processed in GNSS and SFM software, which means that the results are not available for real-time viewing. The system may serve as a starting point for a validated SFM system onto which future engineers may build.

Chapter 2 LITERATURE REVIEW

Photogrammetry has been used successfully across many different fields from archeology to agriculture and even forensics. A coral reef study performed underwater used photogrammetry to create detailed surface maps of living coral environments for further study and is recognized as a cost-efficient tool for ecological surveys [11].

Construction projects have been estimated using photogrammetry as well. While construction progress is monitored, data entry is costly in time and resources. Digitally modeling the current phase of the project of interest offers a visual representation of the structure progress. Additionally, the same UAVs could then be used for safety inspections as well [13]. Law enforcement has attempted to use photogrammetry in evidence analysis using images of shoe imprints in soft surfaces. The image collection process is complicated for some users and could lead to inaccurate models due to poor technical training. However, the ability to use images captured via smartphone with approximately the same quality as evidence collection cameras made the effort to properly train users in evidence collection worthwhile [14]. Archeology has greatly benefited from photogrammetry with SFM to recreate models of artifacts such as statues or jugs for dissemination to the archeological community. The recreations used RGB data mapped to the model for a more accurate representation of the original artifact [15]. Apple orchards are monitored with photogrammetry as well. Tree heights are estimated for informed decisions on watering, pruning and application of fertilizers. Using photogrammetry for agriculture applications greatly reduced the workload on farming communities [16].

Photogrammetry is typically divided into either terrestrial (close range) or aerial [17]. However, the development of photogrammetry in the last decade is mainly because of smaller and cheaper

UAVs [18]. While UAVs were originally created for military applications, the consumer market saw a surge in hobbyist and professional applications starting from the early 2000’s [18].The photogrammetry system designed in this work is also focused on UAV-based SFM.

Best practices for photogrammetry when using UAVs to capture images are reasonably straightforward. The camera may either be placed vertically (nadir) or at an angle (oblique) [19]. Flight may either be controlled autonomously or manually such that there is sufficient overlap among the images and the entire area of interest is covered [19]. Image overlap is recommended to be greater than 80 percent between images [20]. Wind conditions and flight speed must be considered as well. It is generally recommended that flights take place with minimal shadowing for proper color information and while the sun is directly overhead.

SFM evolved from the combination of algorithms used for photogrammetry alongside computer vision methods [18]. As an initial step in a classic SFM algorithm, a sparse point cloud is created using point features captured across multiple images. For proper feature association, it is recommended that each image have approximately 80-90 percent overlap with the previous image. More images are integrated until a sparse point cloud of the 3D model can be estimated. Understandably, this will result in a large amount of images relative to the object being estimated, which requires significant processing time [18].

The sparse point cloud is then scanned for outliers, which are immediately removed. At this stage, there is no absolute location or scaling attributed to the model. The coordinate system in which the 3D model and the camera poses are estimated in can be completely arbitrary [18], although in practice, it may be related to specific camera pose(s). Therefore, neither the camera pose nor the 3D model is linked to a real-world frame.

Additional geo-referenced data is required for tying the model to an applicable coordinate system. There are two forms of georeferencing data. Indirect georeferencing is gathered during the survey by knowing the precise locations of a point feature present in the area being surveyed. These locations are used as GCPs[18]. Direct geo-referencing is performed by integrating the poses of the camera during the flight into SFM. With GCPs and/or camera poses, geo-referencing of the 3D model is accomplished via an adjustment process to correlate the sparse point cloud to a real-world coordinate system [18].

Finally, the geo-referenced point cloud is reconstructed using spatial intersections, gradient and energy minimization algorithms. Based on that, a 3D model with dense point cloud can be created. An optional texture map could also be created by colorizing surface of the model using pixel data extracted from the different images to [18].

Various manufactures and software developers have produced photogrammetry software packages, and some of them are accompanied with small UAVs and sensor packages dedicated to SFM. Popular photogrammetry software applications include Agisoft Metashape, Pix4D, and MicMac [18]. In general, these software packages assume that the user has procured valid, geo-referenced data, with GCPs and/or camera poses, with enough images for sufficient overlap.

There are numerous applications where the placement and survey of GCPs are infeasible or inconvenient [21]. In those cases, SFM relies on the camera pose information for geo-referencing. A camera pose includes both the position and orientation of the camera. It was found that precise position alone can greatly help even if the orientation data is not available [22]. As afore mentioned, precise orientation of a camera relies on a high-quality IMU, which is not always available on small UAVs due to constraints in cost and payload. On the other hand, the equipment needed for precise positioning can be low-cost and low-profile, which will be the main focus of this work.

Some work has been done with RTK combined with photogrammetry to produce rapidly mapped models. Accuracy of the RTK solution as a geo-referenced data point is paramount to the success of the overall solution. Direct referencing allows the system to not rely on pre-arranged and pre-surveyed GCPs. An adjustment must be made for the distance between the camera and receivers to properly use the geo-referenced data [23].

Much work has been done on improving the accuracy of camera position used in SFM, and increasing the accuracy of 3D models as a result [24]. Some preliminary results show that the horizontal accuracy of 1.5 centimeters and vertical accuracy 2.3 centimeters [25]. A minimum of four satellites are required for GNSS solutions, but it is recommended to have at least eight. The redundancy of satellites allows for better accuracy and the mitigation of signal degradation due to faulty data as caused by interferences or other intermittent issues[24]. One of the challenges come from GNSS-challenged environment, which refers to locations with heavy tree cover, significant obstructions and multipath from buildings or signal interference. These locations can have an adverse effect on the overall quality of data and should be avoided if possible [24]. Furthermore, it is more convenient to improve the position accuracy in PPK than in RTK, mainly because satellites with signal degradation can be more reliably detected and removed in post processing.

Another challenge is the synchronization between GNSS and camera. GNSS data and solution (including RTK and PPK) are recorded with respect to the receiver clock or the GNSS clock. In a tightly coupled system, the GNSS receiver and a camera may share the same timing mechanism. In that case, GNSS clock or the receiver clock may be used as a timing source to trigger a camera, such that every image from the camera has a GNSS-synchronized time tag. However, in an open-interface system such as the one discussed in this work, the GNSS receiver may be independent from the timing mechanism of the camera. In other words, the images from the camera are not directly recorded with a GNSS time stamp. Without the time stamp, it would be very difficult to find the camera position at the very moment at which an image is taken. Therefore, without a synchronization mechanism, SFM cannot benefit from the accuracy positioning data. Research has determined that if an unknown time delay between the camera image capture and satellite timing exists, the resulting 3D model will have large errors in geo-registration. For example, it was found in [23] that after three trials, errors were recorded ranging from 0.3 meters to 1.12 meters. These errors were directly attributed to the time delay between image capture and satellite data recording and insufficient adjustment of the focal length and lever arm. Fazeli et. al. discuss that the future improvements could be made by having a faster frame rate taken by the UAV, and reduction of the sensor time synchronization delays [25].

Proprietary systems typically use tightly coupled RTK and camera systems, which guarantees synchronization accuracy. However, these systems do not offer open and interoperable interfaces between RTK/GNSS and the camera. Neither the RTK or the camera can be replaced or upgraded by the user. As a result, the cost of the commercial RTK-camera systems remains very high.

On the other hand, open-source software libraries are now available for live RTK and PPK. RTKlib is a freeware GNSS solution for accurate positioning of outdoor objects. Using a ‘rover’ and ‘base’ receiver, RTKlib can provide real-time and post-processed solutions. It is compatible with a range of GNSS receivers, including low-cost, low-profile receivers such as the U-blox C099-F9P.

For example, Figure 1 displays the computed location of the rover station in reference to a designated base station. The rover is stationary. Without PPK, the computed solution may appear as if it is around in a larger area (shown in yellow path). It corresponds to meter-level accuracy. With PPK, the path converged into a small area (green), which indicates centimeter-level accuracy. Both the rover and base are using U-blox C099-F9P receivers.

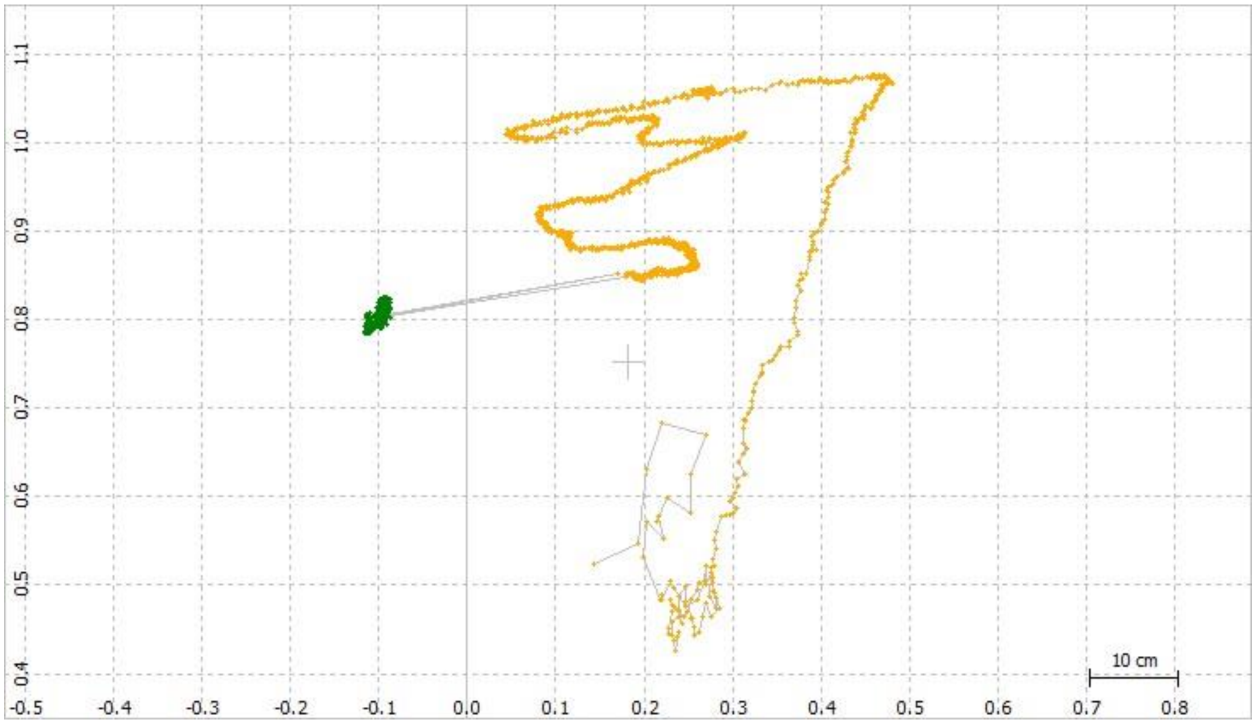


Figure 1: Sample RINEX data analyzed by RTKPLOT in post processing.

Chapter 3 METHOD

A block diagram that includes all the hardware components in the proposed SFM system can be found in Figure 2 and Figure 3. The “rover” is the subsystem that is onboard a small UAV, which includes the GNSS receiver (U-Blox C099-F9P), camera, onboard data recording computer (Raspberry Pi 3B), batteries and the communication circuitry, as shown in Figure 2. The U-Blox C099-F9P receiver uses a small GNSS antenna (in purple) and is powered by a 3.7 V Lithium Polymer battery with no voltage converter required. A PPS LED is connected to the receiver (in green) for use in post processing to synchronize images to GPS time. The GNSS data traffic is passed to the Raspberry Pi 3B computer via a logic level converter. The computer is powered by another 3.7 V Lithium Polymer battery, which requires a voltage converter. The computer has a LED output to indicate the starting time and a pushbutton to initialize the starting time (both in green). The starting LED is used to illustrate to the camera a starting image to be used as the initial image to begin synchronizing positional data to. The camera is an isolated component with a built-in power supply and recording solution and is not directly connected to other components.

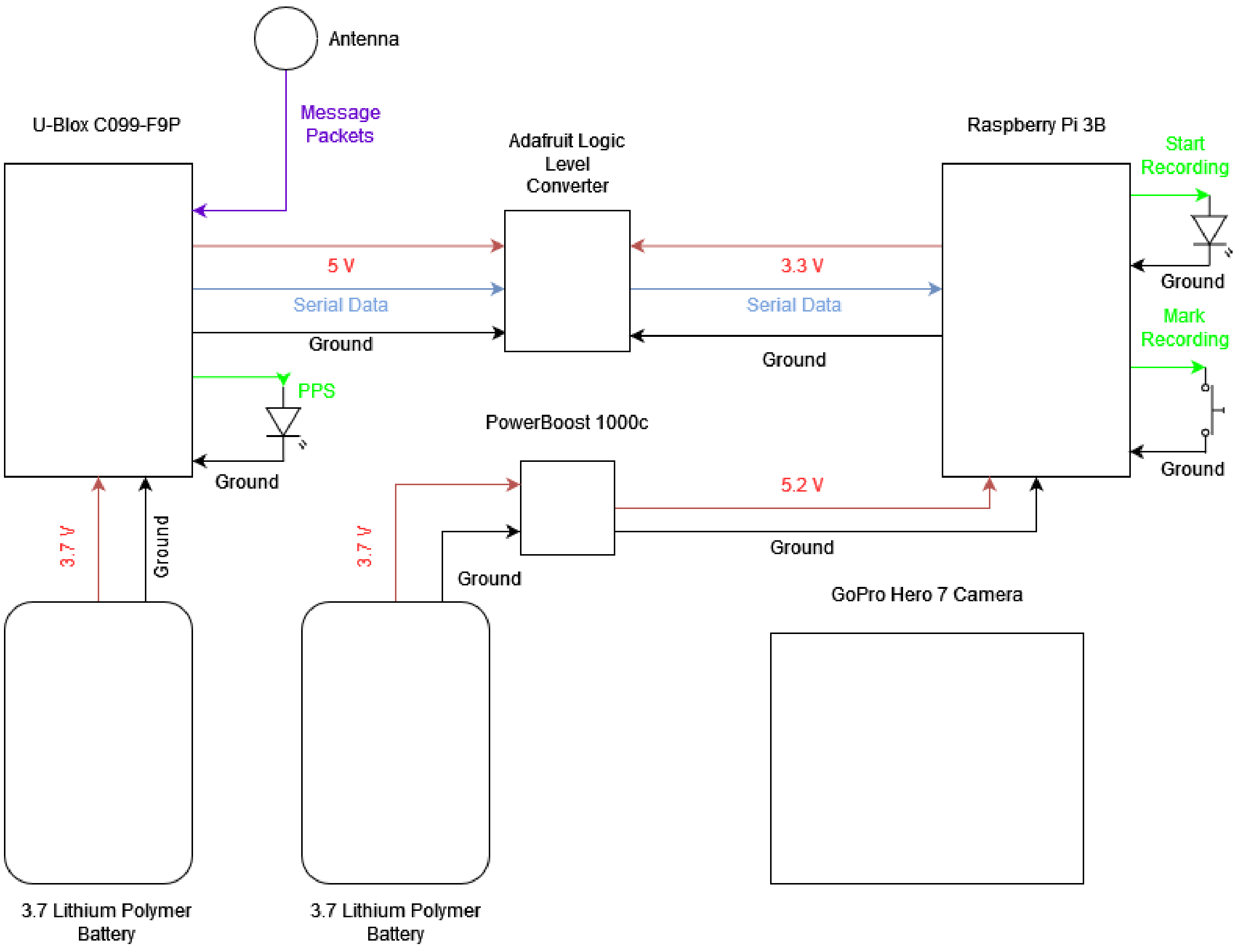


Figure 2: Hardware components of a rover assembly used in SFM.

As a second subsystem, the base station is similar to the rover as shown in Figure 3. The base station will record ground reference data used later for a PPK solution. It also uses a U-Blox C099-F9P including an antenna. Since there is no camera on the base station, no PPS LED is necessary. The receiver is powered by a 3.7 V Lithium Polymer battery as well. The GNSS data traffic is also passed to the Raspberry Pi 3B computer via a logic level converter.

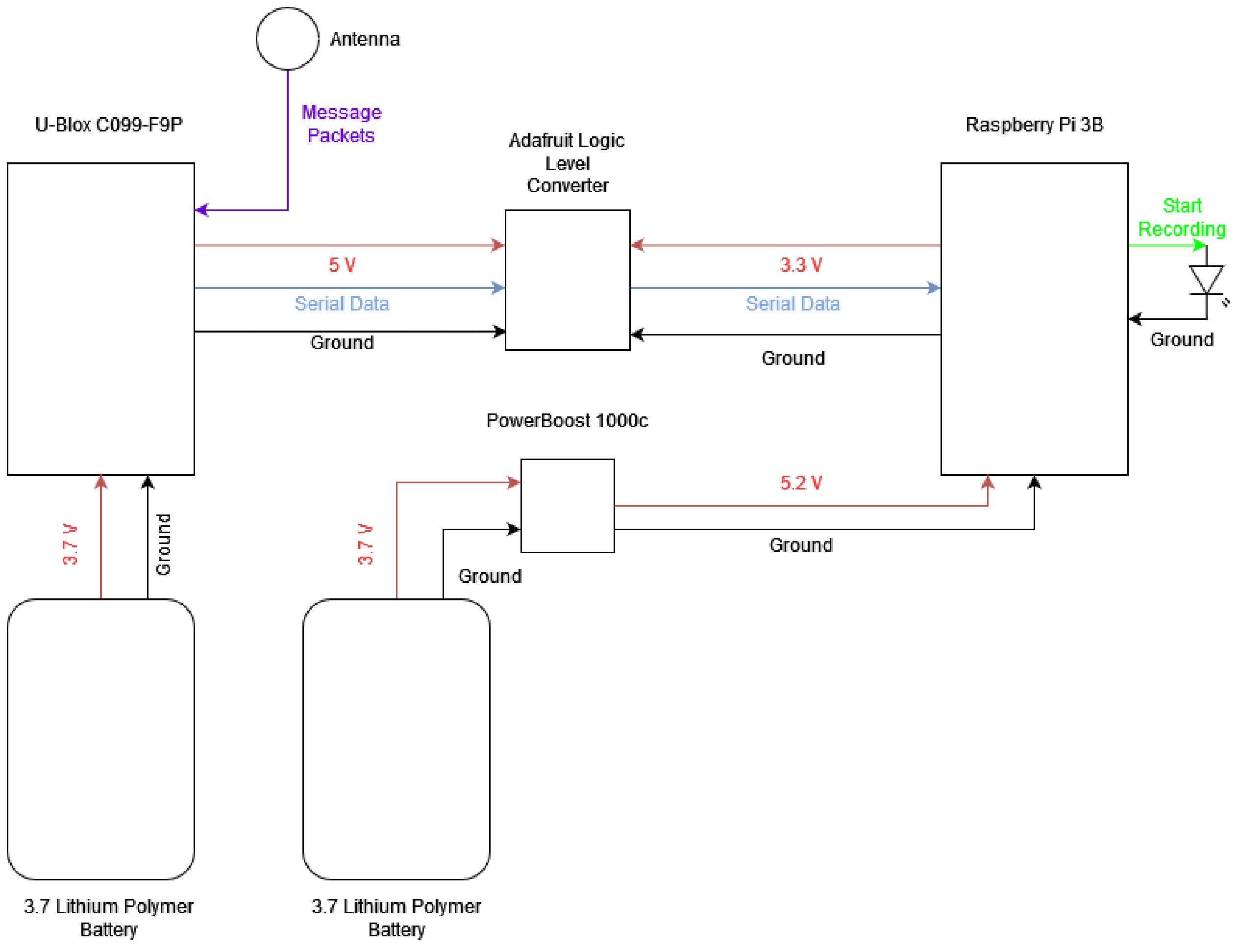


Figure 3: A base station used for ground reference in a SFM solution.

After every flight, the recorded GNSS data and images are post-processed. Therefore, there is no need for the rover and the base station to maintain communications during flight. The GNSS data from both the rover and base station are exported to RTKlib to compute a PPK solution, which precisely measures the relative location of both antennas during the data collection. Naturally, PPK solutions are synchronized to GPS time. In order to integrate them with images, the images also have to be synchronized and time tagged with GPS time. The “start recording” and PPS LEDs are recorded in the images. During synchronization in post processing, images with lit LEDs are identified, which can be precisely time tagged. Based on the time tag and the PPK solution, a reference file is created that associates each time tagged image with a position and accuracy. The images and the reference file are imported into Agisoft Metashape which uses the SFM process to create a three-dimensional model. This model is then validated for accuracy and precision. An overview of this process is shown in Figure 4.

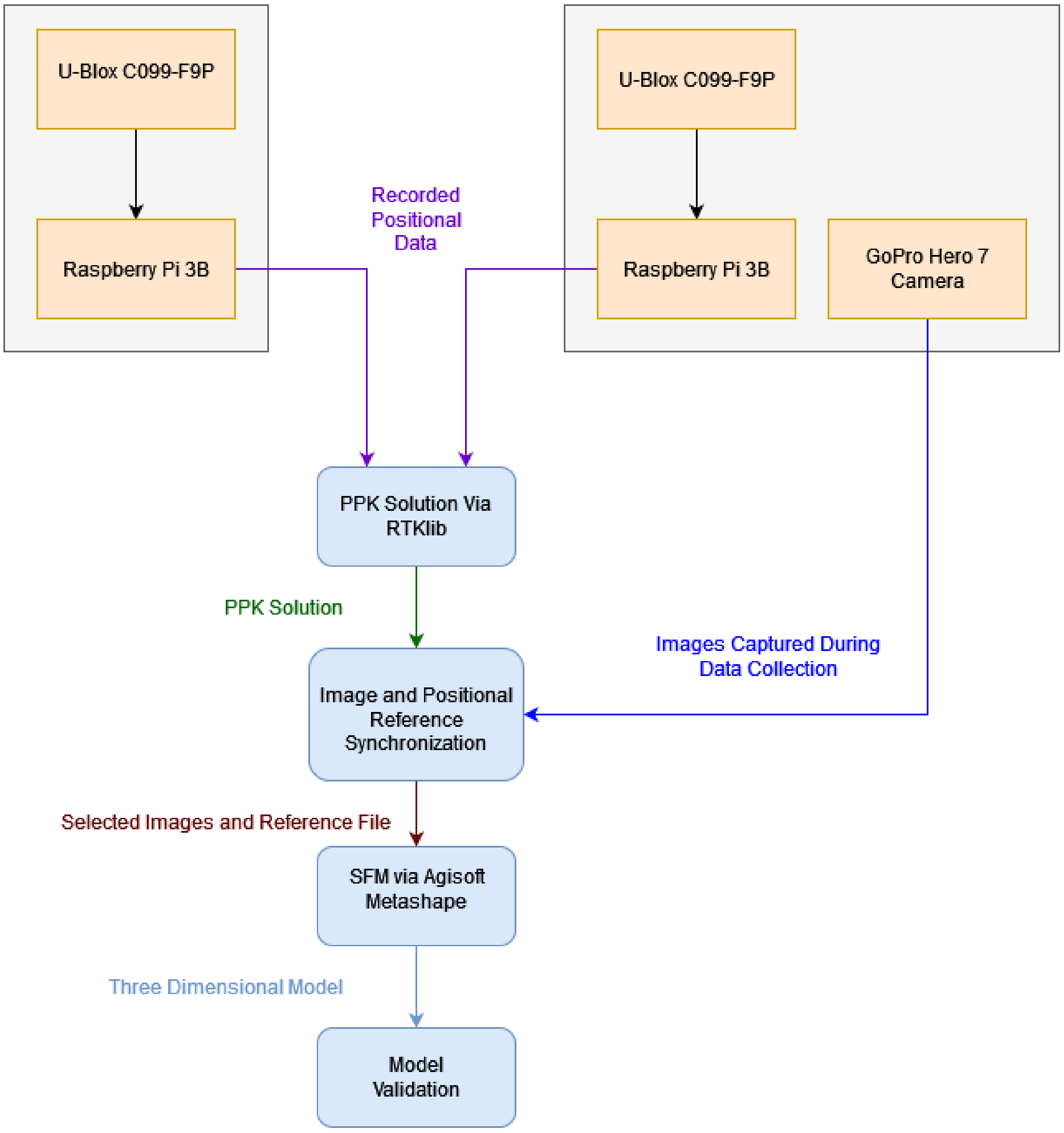


Figure 4: Overview of the SFM model creation workflow as used in this work.

3.1 Camera

In this proposed open-interface system, the camera is required to capture images at a frame rate sufficiently high, such that the time stamp of every image can be precisely matched to the GNSS-based timing of precise position solution. The images must be clear (low blurriness) with a sufficient pixel resolution. The camera is expected to operate on a battery and save images on an internal memory, which can support image capturing at least during a whole flight. Camera images are expected to overlap, as afore mentioned. Therefore, a camera with a fisheye lens is desired for the ability to capture larger areas in an image when compared with standard frame lenses.

The cameras under consideration included the Firefly SplitCam 4k fisheye camera, the GoPro Hero 6 and Hero 7. The Firefly camera was excluded due to an overheating issue, which would have required a cooling system to properly capture at the required frame rate. The GoPro Hero 6 was not selected because of a variable exposure feature and an image stabilization feature that could not be disabled at lower resolution settings. Image stabilization caused the LEDs used for time tagging to vary in position from one image to another, which makes automated synchronization difficult. A GoPro Hero 7 was chosen for this project. It can continuously record for half an hour, at a frame rate of 240 frames per second (fps). The Hero 7 also has the capability of locking the shutter speed and disabling the adaptive exposure. A constant shutter speed is desired such that the time delay of images due to exposure time is constant. Adaptive exposure also causes uncertainty in timing. With that feature, the camera would record a staggering of images in groups of eight to ten images, with a pause in between each image group for recording the images to a memory card. It would also make timing unnecessarily complicated.

With these settings disabled, the GoPro Hero 7 camera can record evenly at 240 fps with a constant interval between images. It can be verified using a bank of LEDs. As shown in Figure 5, sixteen LEDs were organized in a row and controlled with an Arduino Mega 2560.

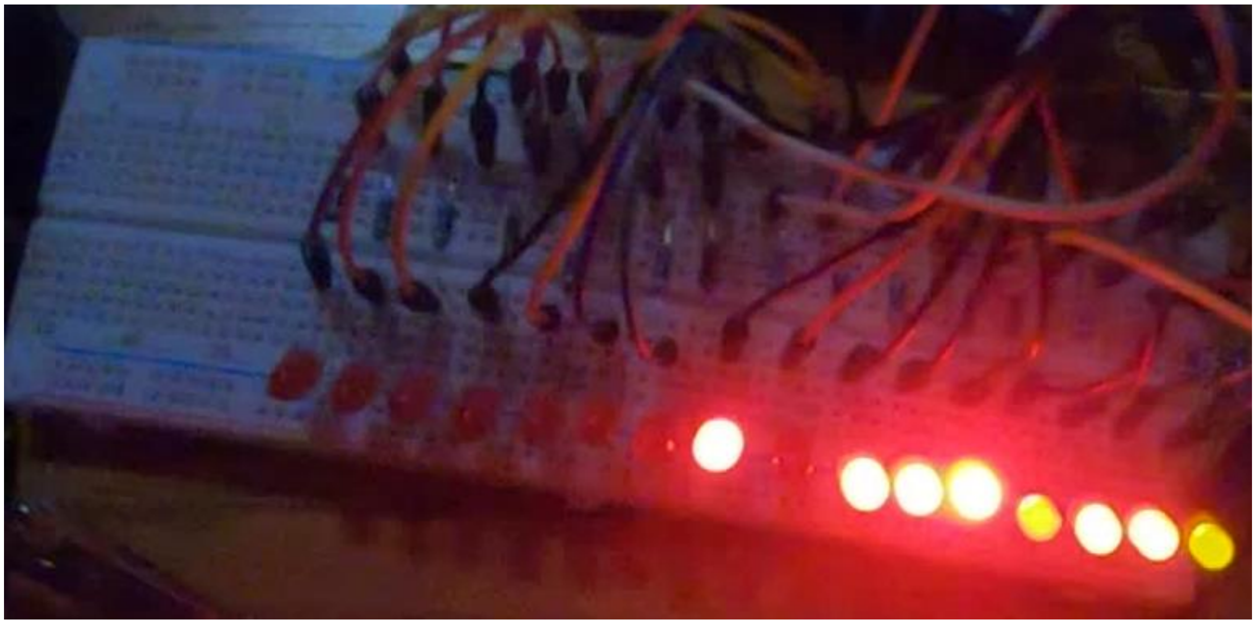


Figure 5: LED pattern used to validate camera capture rate.

An Arduino program was developed to display a binary representation of decimal numbers with these LEDs. The number to be displayed was increased from one to one thousand, with a millisecond between each step. An oscilloscope was used to verify that the accuracy of timing in the LEDs for each step.

A separate Python program was developed to retrieve the number in the images based on the brightness of LEDs. The number was then compared across a series of images to determine whether the timing of images was evenly distributed at the specified frame rate.

However, during the initial testing, it was discovered that due to the transition between LED status, the number could not always be corrected decoded from images. As shown in Figure 6, the millisecond number did not increase linearly as expected.

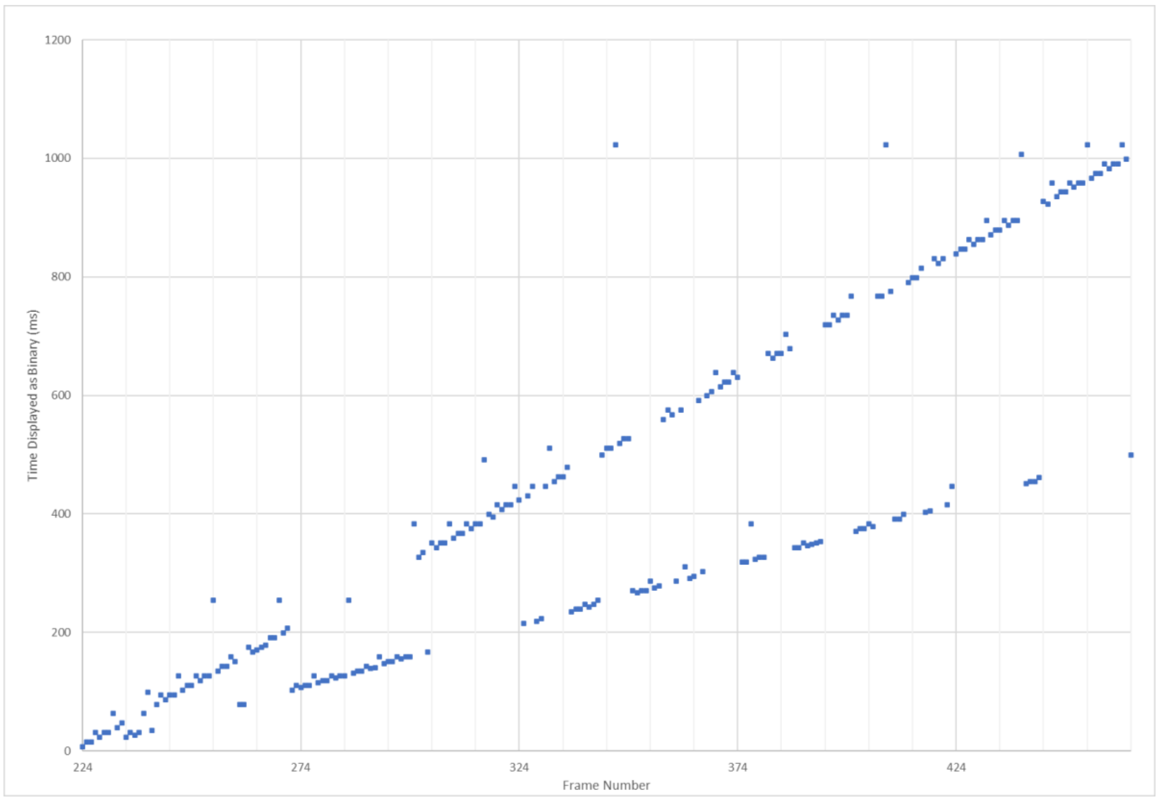


Figure 6: Counting errors present when using binary counting.

As a result, a direct binary representation of decimal numbers cannot be used in this test. Instead, ‘Gray Coding’, or inverse binary, was implemented. Gray coding is a method of representing a numeric value with only a single LED changing states from any one value to the next higher value [27].

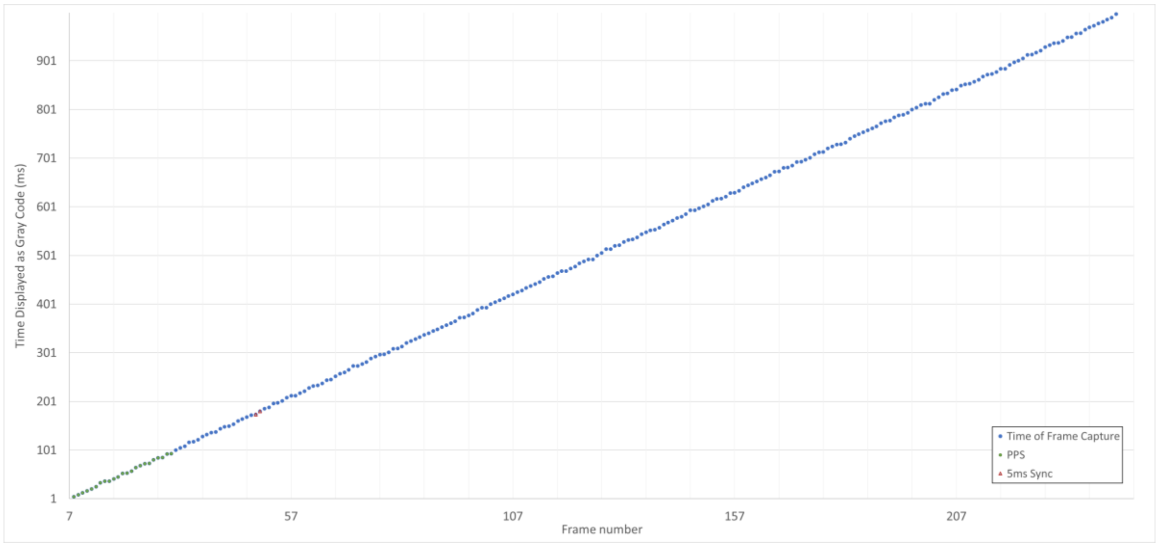


Figure 7: Proper sequential counting after modification to use ‘Gray Coding’.

After the system was updated to display and decode decimal numbers with gray coding, the numbers retrieved from continuous images appear in a linearly incrementing pattern, as shown in Figure 7. It is further verified that there is an approximately four-millisecond interval between consecutive images in a zoomed in view in Figure 8. Therefore, it was concluded that the images are consistently and evenly timed at 240 FPS with the Hero 7 camera.

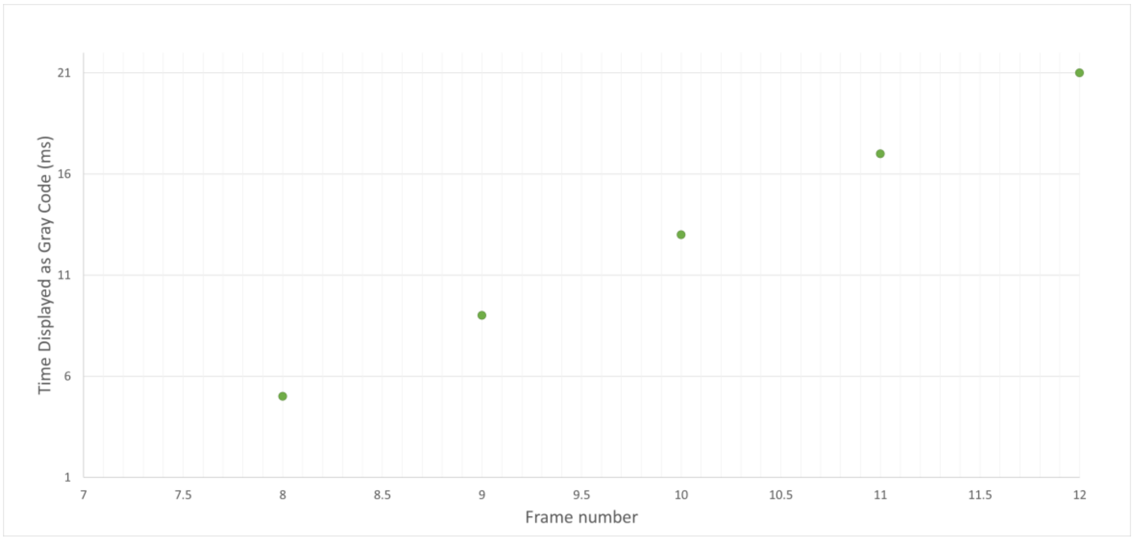
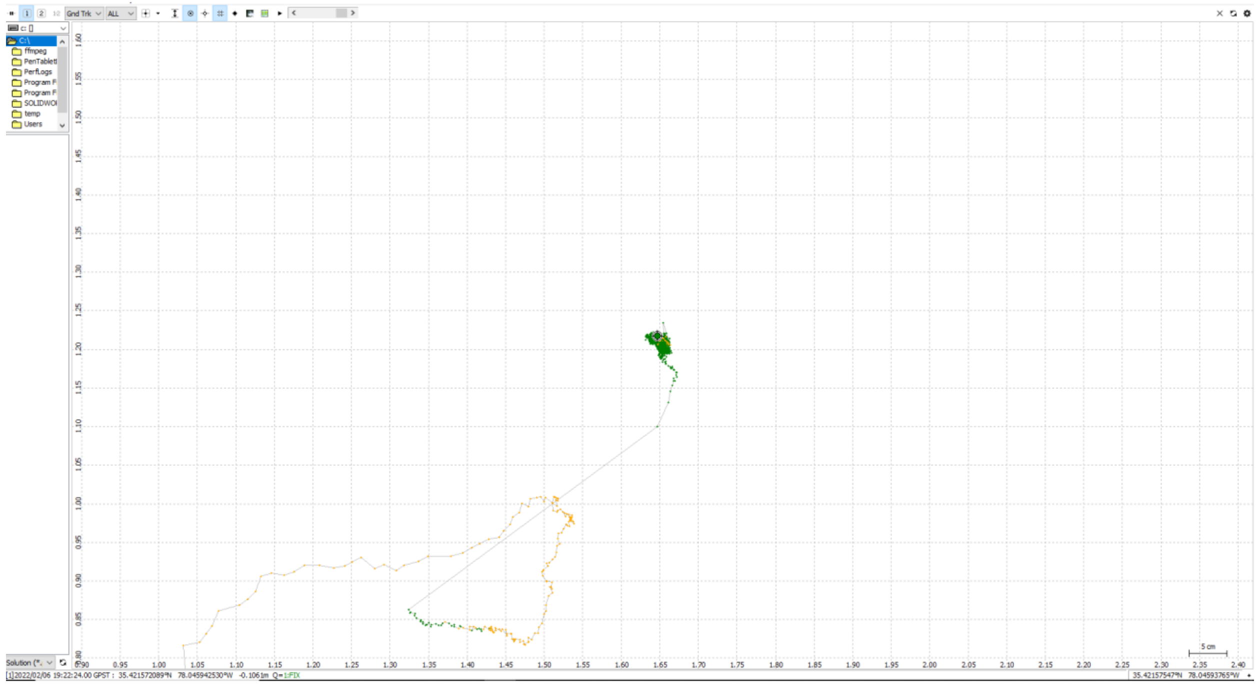


Figure 8: Consistent spacing between frames with no recording lag present.

3.2 GNSS Receiver

The U-blox F9P is a low-cost GNSS receiver chip that is capable of RTK and data recording, which is selected in this project. It was embedded on the U-blox C099-F9P development board, which allows easy access to multiple GNSS data streams, and the configuration of data rate, message types, formats, and other relevant settings. The C099-F9P allows for the message length to be shortened by filtering out unnecessary packets and adjusting the data transfer speeds to be the maximum allowed by the recording device. This reduces transmission time and allows time for the recording device to save the inbound message traffic before the next set of message packets arrives. In this project, the C099-F9P board is configured as a standalone receiver which relays the message packets to two separate data ports, either of which are acceptable for use by the recording device.

To verify RTK-compatibility, two separate receivers are configured and attached to a Windows PC via USB. Raw GNSS data were recorded from both receivers in the U-Blox proprietary data format. Both data files are then processed with RTKLib, creating a PPK solution of the designated ‘rover’ referenced to the designated ‘base’, which can be observed in Figure 9.



```
[1]2022/02/06 19:22:24.00 GPST : 35.421572089°N 78.045942530°W -0.1061m Q=1:FIX
35.42157547°N 78.04593765°W
```

Figure 9: Resulting RTK solution from positional data recorded.

Similar to the sample shown in Figure 1, the part of the solution shown in green is from a PPK solution, which reaches centimeter-level accuracy.

3.3 Data Recording

Data recording during a flight test could not rely on a large Windows computer. Instead, a compact recording system was developed. Although the U-blox C099-F9P has an on-board hardware interface for a micro-SD memory card, there is no software interface provided by the manufacturer to enable recording locally. Therefore, the data had to be recorded off board in this project.

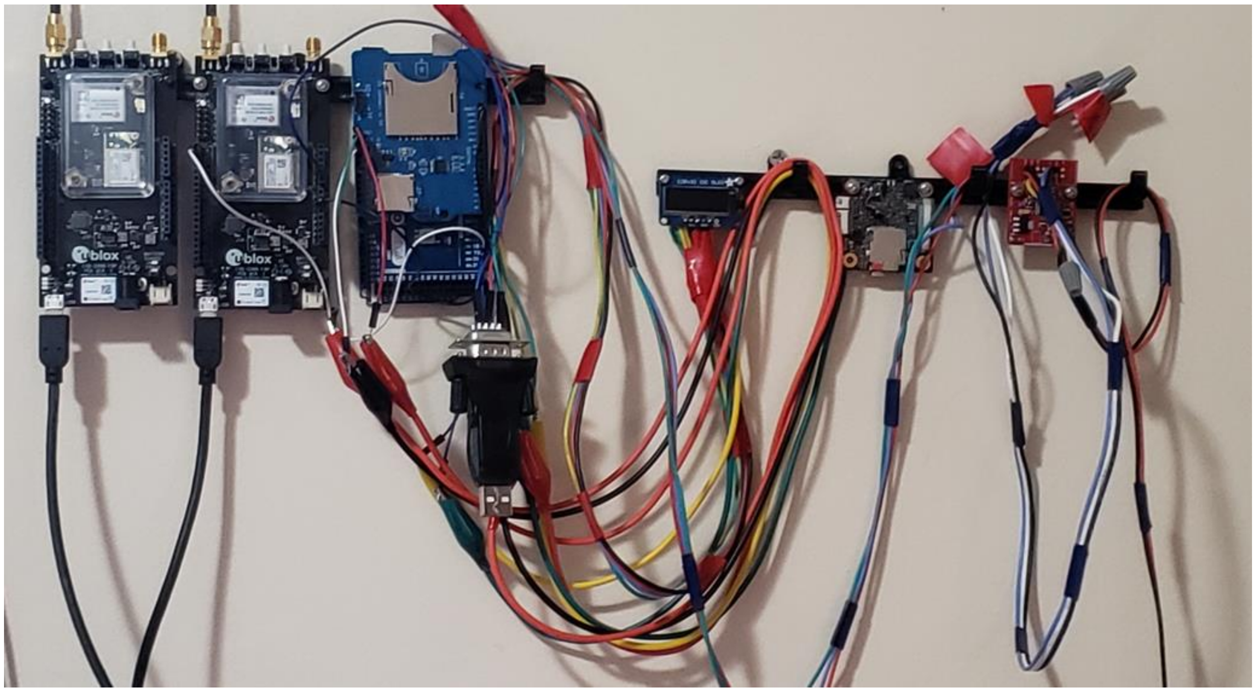
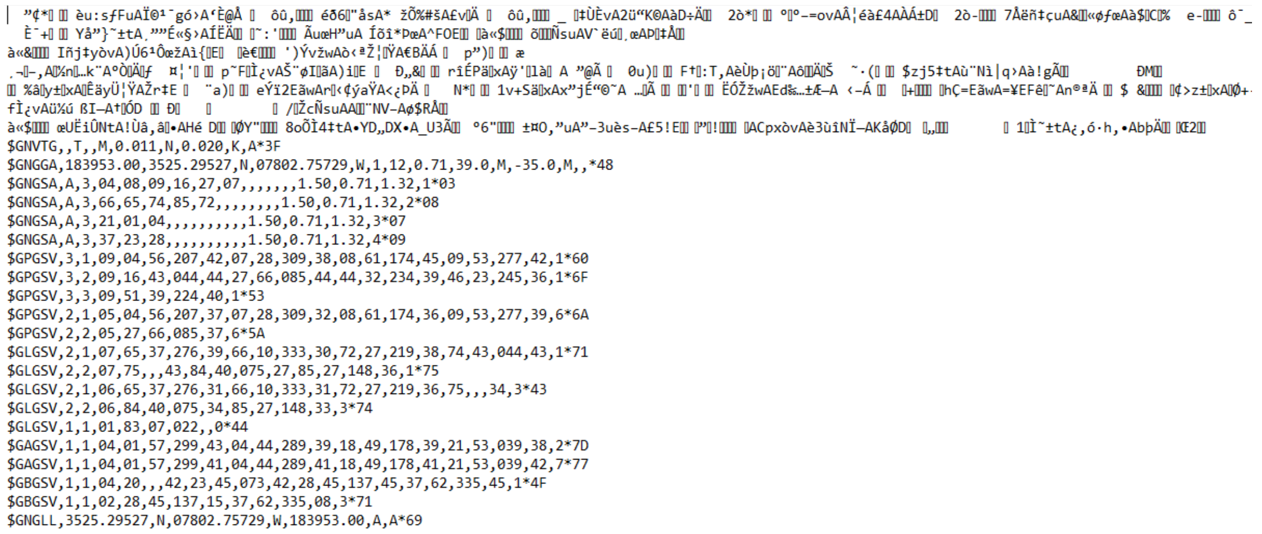


Figure 10: Testing of various display and recording methods of positional data.

Simple serial data loggers and microcontrollers seen in Figure 10 could not support the maximum data speed required. Instead, a Raspberry Pi 3B computer is able to receive the data serially at 115200 baud, and still has sufficient processing power to pre-process the messages. It is relatively small, low power and cost effective. A Python code was developed for this computer to manage the serial connection, receive, decode and record the data from the C099-F9P receiver. It records in both binary and text formats. Figure 11 provides an example of raw data recorded in Raspberry Pi 3B, converted into text. The binary data files follow the Ublox proprietary format, and can be directly used in RTKlib.



```
| ”¢*□ □□ èu:sƒFuAÏ©¹¯gó›A‘È@Å □  ôû,□□□ éð6□"åsA* žÕ%#šA£v□Ä □  ôû,□□□ _ □‡ÙÈvA2ü“K©AàD÷Ä□□  2ò*□ □□ º□º-=ovAÂ¦éà£4AÀÁ±D□  2ò-□□□ 7Åëñ‡çuA&□«øƒœAà$□C□%  e-□□□ ô¯_
  È¯+□ □□ Yå”}˜±tA¸””É«§›AÍËÄ□□ □˜:'□□□ ÃuœH”uA Íõî*ÞœA^FOE□□ □à«$□□□ õ□□ÑsuAV`ëú□¸œAÞ□‡Å□□
à«&□□□ Iñj‡yòvA)Ú6¹ÔœžAì{□E□  □è€□□□ ')ÝvžwAò‹ªŽ¦□ŸA€BÄÁ □  p”)□ □□ æ
¸¬□–,A□½n□…k¨AºÒ□Ä□ƒ  ¤¦'□ □□ p˜F□Ì¿vAŠ¨øI□ãA)i□E □  Ð„&□ □□ rîÉPä□xAÿ'□là□ A ”@Ã □  0u)□ □□ F†□:T,AèÙþ¡ö□¨Aô□□Ä□Š  ˜·(□ □□ $zj5‡tAù¨Nì|q›Aà!gÃ□□         ÐM□□
□□ %â□y±□xA□ÊäyÜ¦ŸAŽr‡E □  ¨a)□ □□ eÝï2EãwAr□‹¢ýaŸA<¿ÞÄ □   N*□ □□ 1v+Sä□xAx”jÉ“©˜A …□Ã □□ □□'□ □□ ËÕŽžwAEd‱…±Æ–A ‹–Á □□  □+□□□ □hÇ=EãwA=¥EFê□˜An®ªÄ □□ $ &□□□ □¢>z±□xA□Ø+·
fÌ¿vAü½ú ßI–A†□ÓD □□ Ð□    □          □ /□ŽcÑsuAA□□¨NV–Aø$RÅ□□
à«$□□□ œUËiÛNtA!Ùâ,â□•AHé D□□ □ØY"□□□ 8oÕÌ4‡tA•YD„DX•A_U3Ã□□  º6"□□□ ±¤O,”uA”-3uès–A£5!E□□ □”□!□□□ □ACpxòvAè3ùîNÏ–AKåØD□  □„□□       □ 1□Ì˜±tA¿,ó·h,•AbþÄ□□ □Œ2□□
$GNVTG,,T,,M,0.011,N,0.020,K,A*3F
$GNGGA,183953.00,3525.29527,N,07802.75729,W,1,12,0.71,39.0,M,-35.0,M,,*48
$GNGSA,A,3,04,08,09,16,27,07,,,,,,,1.50,0.71,1.32,1*03
$GNGSA,A,3,66,65,74,85,72,,,,,,,,1.50,0.71,1.32,2*08
$GNGSA,A,3,21,01,04,,,,,,,,,,1.50,0.71,1.32,3*07
$GNGSA,A,3,37,23,28,,,,,,,,,,1.50,0.71,1.32,4*09
$GPGSV,3,1,09,04,56,207,42,07,28,309,38,08,61,174,45,09,53,277,42,1*60
$GPGSV,3,2,09,16,43,044,44,27,66,085,44,44,32,234,39,46,23,245,36,1*6F
$GPGSV,3,3,09,51,39,224,40,1*53
$GPGSV,2,1,05,04,56,207,37,07,28,309,32,08,61,174,36,09,53,277,39,6*6A
$GPGSV,2,2,05,27,66,085,37,6*5A
$GLGSV,2,1,07,65,37,276,39,66,10,333,30,72,27,219,38,74,43,044,43,1*71
$GLGSV,2,2,07,75,,,43,84,40,075,27,85,27,148,36,1*75
$GLGSV,2,1,06,65,37,276,31,66,10,333,31,72,27,219,36,75,,,34,3*43
$GLGSV,2,2,06,84,40,075,34,85,27,148,33,3*74
$GLGSV,1,1,01,83,07,022,,0*44
$GAGSV,1,1,04,01,57,299,43,04,44,289,39,18,49,178,39,21,53,039,38,2*7D
$GAGSV,1,1,04,01,57,299,41,04,44,289,41,18,49,178,41,21,53,039,42,7*77
$GBGSV,1,1,04,20,,,42,23,45,073,42,28,45,137,45,37,62,335,45,1*4F
$GBGSV,1,1,02,28,45,137,15,37,62,335,08,3*71
$GNGLL,3525.29527,N,07802.75729,W,183953.00,A,A*69
```

Figure 11: Sample positional data recorded by the Raspberry Pi 3B in plain text.

Logic for recording positional data and post-processing was written in Python 3. Recording positional data was saved as scripts on Raspberry Pi 3Bs and initialized via run commands embedded in the rc.local file. Post processing code was run via the PyCharm integrated development environment (IDE). All code has been included in the referenced appendices.

The code used in the rover handles GNSS data recording and activates the indicator LEDs. After initializing the required libraries and variables, an attached LED indicates that the rover is recording GNSS data by blinking four times. The rover will standby in a looped recording pattern until the attached button is pressed. When the button is pressed, the rover will then create timestamped files used in post processing to indicate when the GNSS data related to the images begins.

The logic in the base station records GNSS data and controls indicators displayed to the user. After initialization, the base station logic opens log files in binary and text format and begins to record GNSS data from the U-blox C099-F9P. A logic level converter is used similar to the rover to adjust logic level voltages. The attached recording LED indicator alternates states from lit to unlit with each GNSS data packet. This serves as an easy indicator of correct operation by the base station while recording. The GNSS data packets are counted and after each set of ten the base station will close and reopen the files in append mode. This prevents data loss during recording when the base station is powered down.

3.4 Power Supply

The proposed photogrammetry system is a standalone module that can be attached to an arbitrary airframe. Therefore, it cannot rely on any external power supply, including UAV batteries. Furthermore, the GNSS receiver, camera and onboard computers all require different DC voltage levels. The Raspberry Pi 3B requires 5.2V DC and the U-blox C099-F9P development board accepts a JST connection supplied with 3.7V. Two methods were considered to solve this problem. A 12V Lithium Polymer battery could be used, which can be stepped down to lower voltage, such as 5.2V and 3.7V needed by the computer and GNSS respectively, via DC-DC voltage converters. Alternatively, two 3.7V Lithium Polymer batteries may be used, and one of them can be stepped up to provide the 5.2V required by the computer. The second solution also has the benefit of having much less weight because of the smaller batteries. It is selected for this system, together with an Adafruit Powerboost 1000c voltage step up converter.

The power supply system was verified. The system was able to continuously run for a half hour with no significant drain on the batteries. The flight time required for a complete scan of a set of targets is estimated around ten minutes of flight time. Therefore, the designed power supply system is more than capable of supporting these flights. If additional flights are needed that exceed the power capabilities of two batteries, it is very convenient to replace the batteries before each flight.

3.5 Timing Synchronization

With an open interface between the camera and the GNSS receiver, the image time tag cannot be directly recorded as in a closed proprietary system. Instead, approaches to reflect time tag into images are explored. The first option considered in this project was to display the GPS time in plain text with an OLED matrix screen, which will be captured by the camera directly. Although it could clearly provide a GPS time tag for every image, it raises challenges in post processing. Since the text appears blurred in the images, it was difficult to read the time tag.

Alternatively, a bank of LEDs can be used to display binary time. For a six-digit time tag, 17 LEDs would be required. This was achieved with a curved bar placed underneath the camera, as shown in Figure 12 with the LEDs protruding into the visible range of the camera.

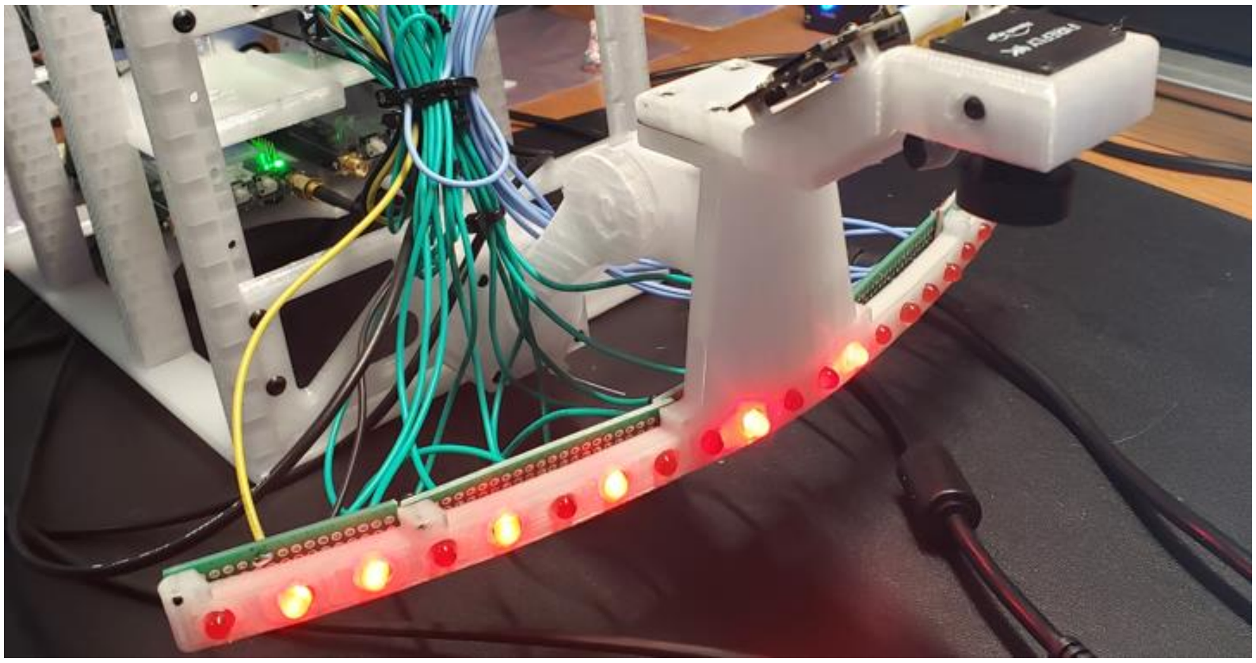


Figure 12: Original method to display timing information within captured images.

Because of the fisheye lens, the LED bar was curved so that it would appear to be straight along the bottom of the image. An additional LED is required to synchronize the time seen with the instant that the C099-F9P receiver obtained the GNSS data. An additional Arduino Mega 2560 was used to control the conversion from GPS time seen on the C099-F9P to a binary representation. However, it was noticed after initial testing that the LEDs were crowded in a proximity, which could cause false positive readings when a LED reflects light from neighbors. Therefore, this option was not feasible for UAV flight.

A much simplified solution is implemented. It consists of only two LEDs. One LED is connected to the C099-F9P and outputs a time pulse, or pulse per second (PPS). This LED is triggered by the beginning of a GPS second, and remains high for one hundred milliseconds. It will be a logic low for the subsequent 900 milliseconds until the next second. With the LED installed in a constant position, the transition between low and high can be easily monitored during post processing. The first image captured with a high indicates the closest known instant to the beginning of a GPS second. With images at 240 fps, this approach will only introduce a small timing error.

The secondary LED is connected to a General Purpose Input/Output (GPIO) port on the Raspberry Pi 3B. A pushbutton is connected to a separate GPIO port as well. When recording is ready to begin, the user presses and holds the button until the secondary LED lights up. After two seconds the LED will extinguish. The LED will light up again when the next GNSS data packet is received. The presence of the lit LED for the second time after the pushbutton is pressed indicates that the next GNSS packet will begin the recording process. The time corresponding to that next GNSS packet is written to a separate file and is used during the image selection process. After the following GNSS packet is received, the LED extinguishes, and is no longer used in the data collection process. The LEDs used in this work are seen in Figure 13 through Figure 15.

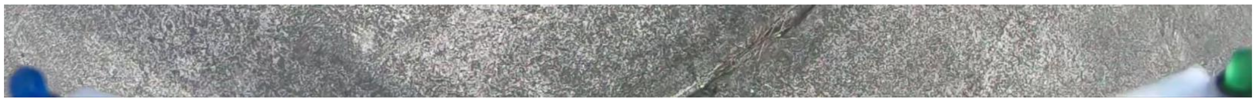


Figure 13: Original LED for PPS and Start Recording states as seen by camera.

After initial trials, it was observed when rotating the photogrammetry module during data capture caused a reflection of sunlight to be captured on the dome of the LEDs. To correct this issue, the dome-style LEDs originally used were replaced with flat surface mounted LEDs.

Figure 14: Flat surface mounted LEDs used to reduce reflected light errors.

While the surface mounted LEDs minimized the glare, there was still enough sunlight reflecting to occasionally cause false positive lit conditions during post processing. At this stage, additional domes were manufactured with a 3D printer with slits facing the camera lens were created.

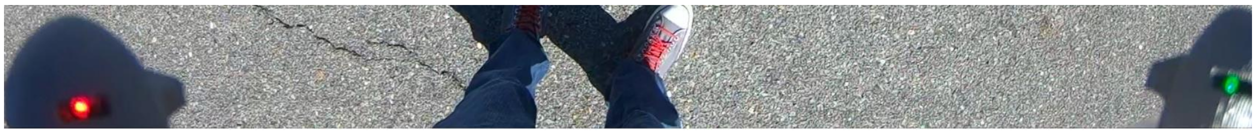


Figure 15: Dome coverings isolating the status LEDs from light interference.

The domes blocked the majority of the sunlight from reaching the LED surface yet still allowed the camera to clearly observe the LED in a static position. The LED dome installation prevented further false positive high indications from appearing during post processing.

Python 3 code was developed to handle the synchronization phase of post processing. This alleviated the tedious task and substantial amount of time required to validate the positional data, format the files, select the images, and log each image with a positional datapoint. For the initial verification of the PPK solution, the code in Appendix C is used. A flow diagram of the process used for verification of the PPK solution is shown in Figure 16. Flow verification scans the PPK solution for missing entries (indicating lost data or GNSS challenged environments) and inserts blank entries with the correct timestamp into the log. This ensures the correct GNSS data is assigned to the correct image in post processing.

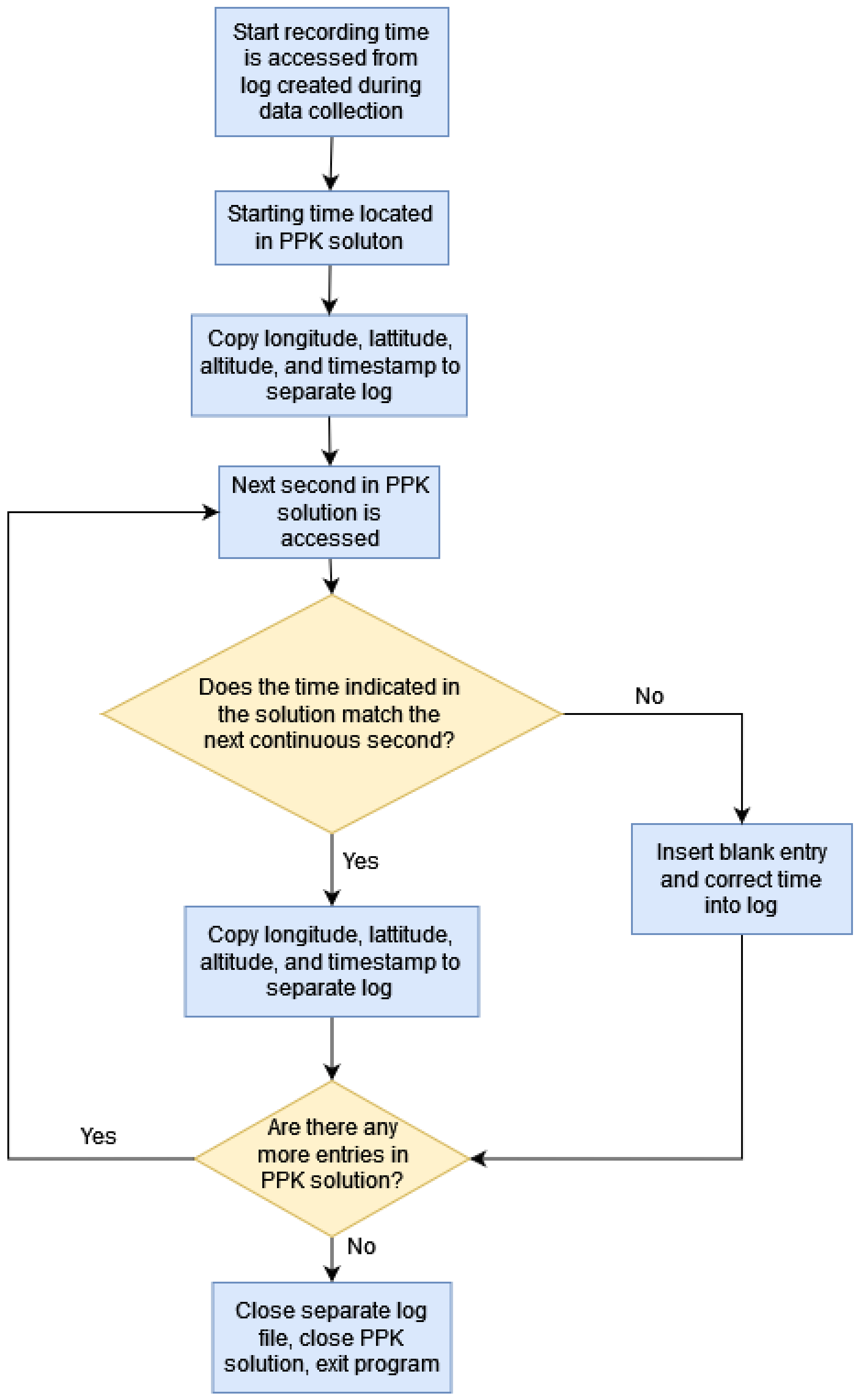

Start recording time is accessed from log created during data collection
Starting time located in PPK soluton
Copy longitude, lattitude, altitude, and timestamp to separate log
Next second in PPK solution is accessed
Does the time indicated in the solution match the next continuous second?
No
Insert blank entry and correct time into log
Yes
Copy longitude, lattitude, altitude, and timestamp to separate log
Yes
Are there any more entries in PPK solution?
No
Close separate log file, close PPK solution, exit program

Figure 16: Flow of record validation for the PPK solution.

The synchronization step ties GNSS data to images that are taken closest to the instant that the GNSS data was received by the U-Blox C099-F9P receiver. It begins during data collection, in which the PPS LED illuminates within the field of view of the camera whenever new GNSS data is received. After the initial bootup phase of the photogrammetry module, a button is pressed indicating that the operator is ready to begin data collection. The start recording LED is lit briefly, acknowledging the start recording command, then flashes once more to indicate the first GNSS data packet that recording begins with. The PPS LED continues to flash when new data is received for the duration of data collection. During post processing, the first image in which the PPS LED is lit after the start recording LED has been lit is copied to a separate folder, along with each subsequent image in which the PPS LED is first detected as being lit. A log is created with the name of these images, and by stepping through the verified PPK solution, GNSS data is added to the log for each image. This synchronizes images in which the GNSS data was received to the location of the UAV in that instant. The synchronization process is summarized by Figure 17.

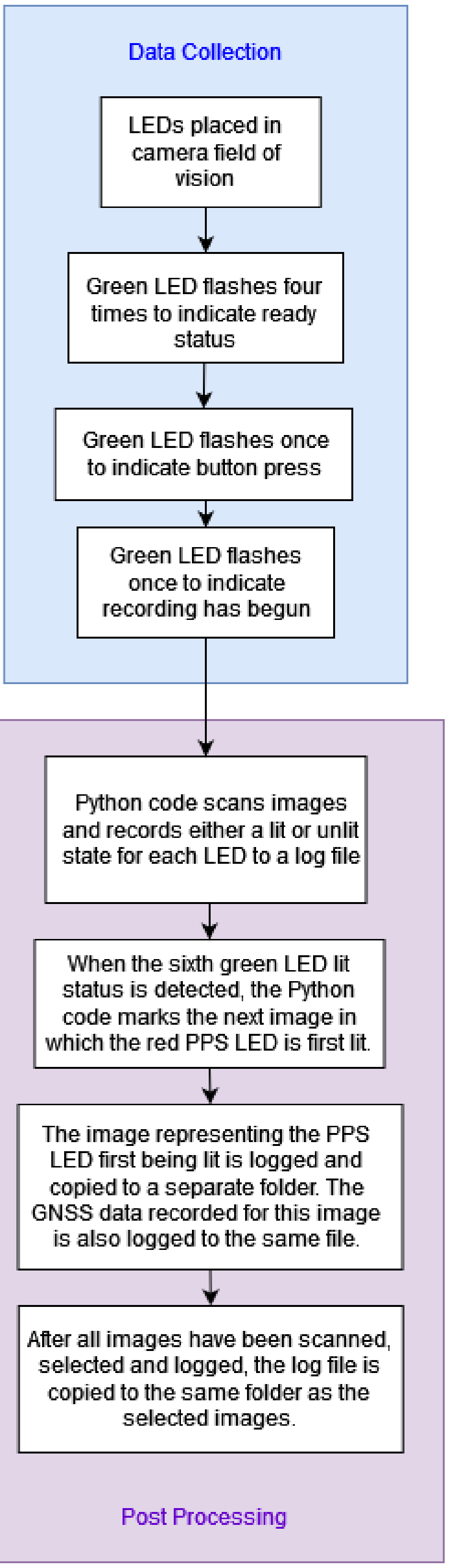

Data Collection
LEDs placed in camera field of vision
Green LED flashes four times to indicate ready status
Green LED flashes once to indicate button press
Green LED flashes once to indicate recording has begun
Python code scans images and records either a lit or unlit state for each LED to a log file
When the sixth green LED lit status is detected, the Python code marks the next image in which the red PPS LED is first lit.
The image representing the PPS LED first being lit is logged and copied to a separate folder. The GNSS data recorded for this image is also logged to the same file.
After all images have been scanned, selected and logged, the log file is copied to the same folder as the selected images.
Post Processing

Figure 17: The synchronization step as handled during data collection and post processing.

The synchronization step is verified by checking the timestamps of the recording and by graphing the LED status of all of the images gathered during the data collection phase. The PPK solution should span for a period of time greater than that of the time taken to complete one data collection event. If the timespan is significantly shorter, then substantial data loss has occurred, and the synchronization step is unable to be completed.

The status of the LEDs used in data collection also is an indicator of the success of the synchronization phase. If the LED status is graphed, a clear repeating pattern of the state of the recording and PPS LEDs is displayed. Distortion of this pattern indicates either errors in LED status detection or light interference detected by the camera in the general location of the LEDs. By scanning the graph visually, pattern distortion is easily detected and the corresponding data points may be referenced to indicate the time of recording in which the errors occurred. The LED detection thresholds may then be modified in the Python code, or the data collection even may be repeated if the errors cannot be mitigated.

3.6 SFM Software

The main focus of this thesis is on system design and hardware implementation. Although there are open source and free software for SFM functionalities, Agisoft Metashape was selected since an educational license was available in the college. It may be replaced with open source SFM in the future. Agisoft possesses the ability to take images and use external geo-reference sources to tie a variety of formatted positional data to images. The images are then used to create point clouds and mesh a model of the feature. While processing intensive, the workflow and batch processing available allows the different stages of model creation to be scheduled and ran automatically without intervention. Pix4D is an alternative capable of geo-referenced source importing.

An early calibration step of the SFM process uses a chessboard displayed by the Agisoft Metashape program. The images captured of a calibration chessboard can be sorted into a chunk separate from the images captured during flight. The lens calibration can be completed to accommodate any distortions caused by the fisheye lens [28]. The results of the calibration of the fisheye lens used in the Hero 7 camera may be shown in Figure 18.

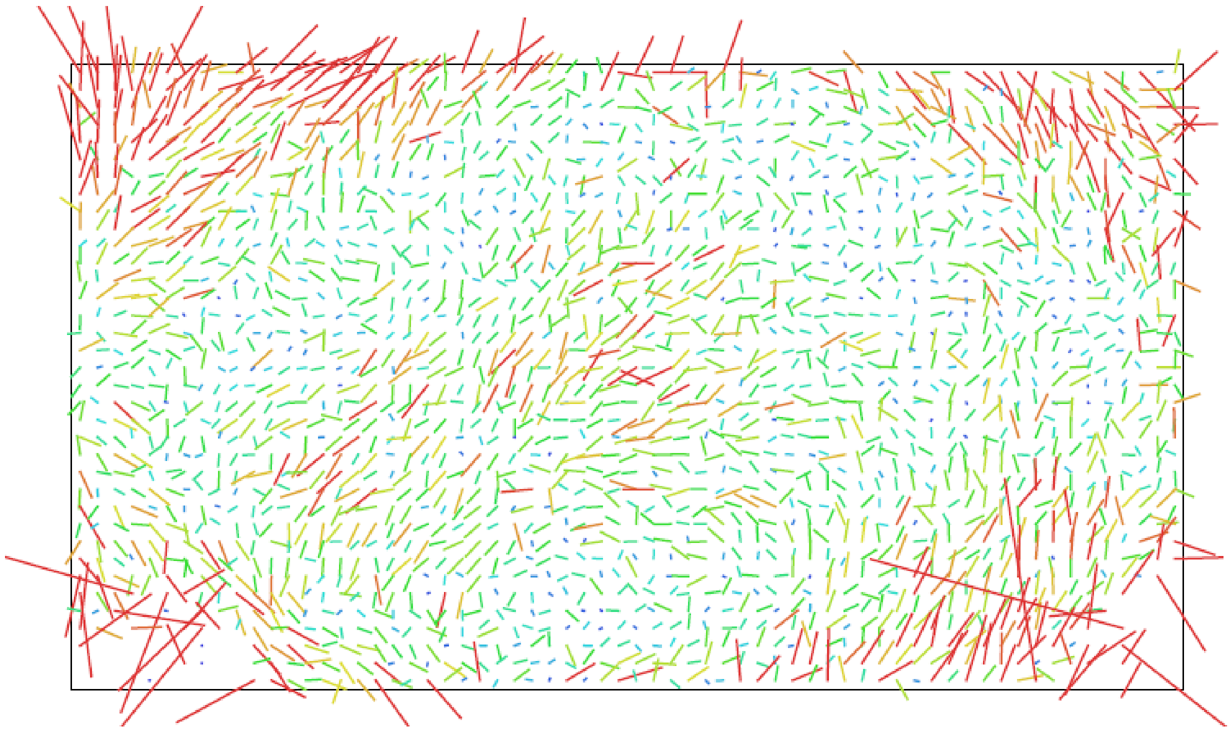


Figure 18: Lens calibration as indicated by Agisoft Metashape Software.

Test subjects were created using painted boxes to test the functionality of the software. These boxes were painted with grids of different sizes. 3D models of these subjects were successfully created and verified against the actual dimensions. A texturized model of the painted box used for initial testing is shown in Figure 19.

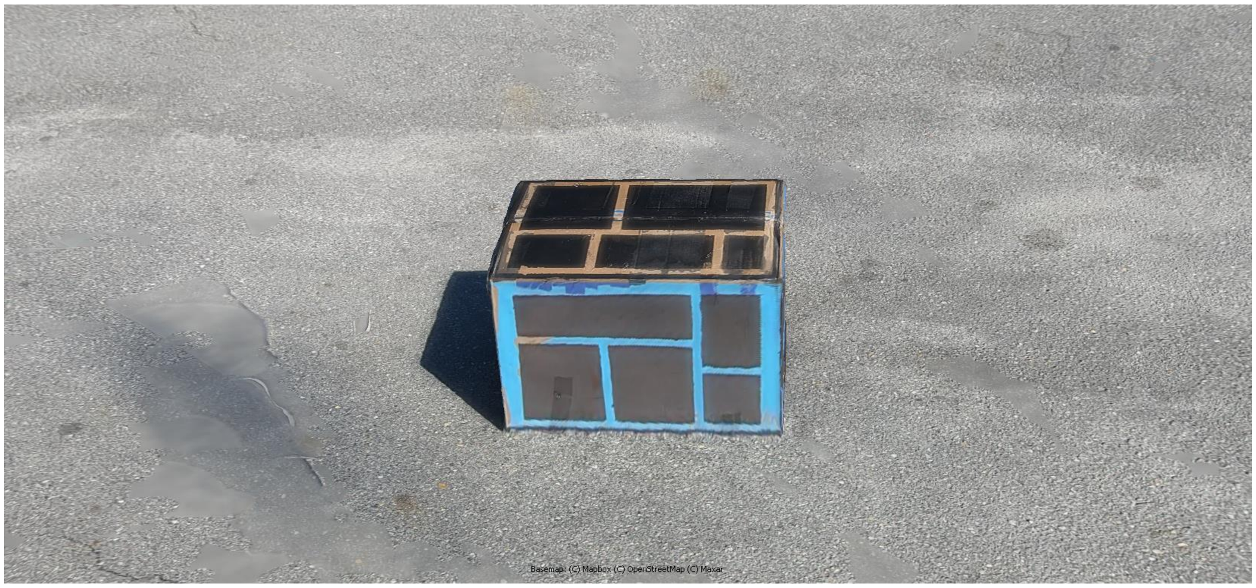


Figure 19: Initial test subject for system operation tests: a cardboard box with a painted grid.

However, subjects with glass or severely reflective materials on the surface were more prone to errors in SFM.

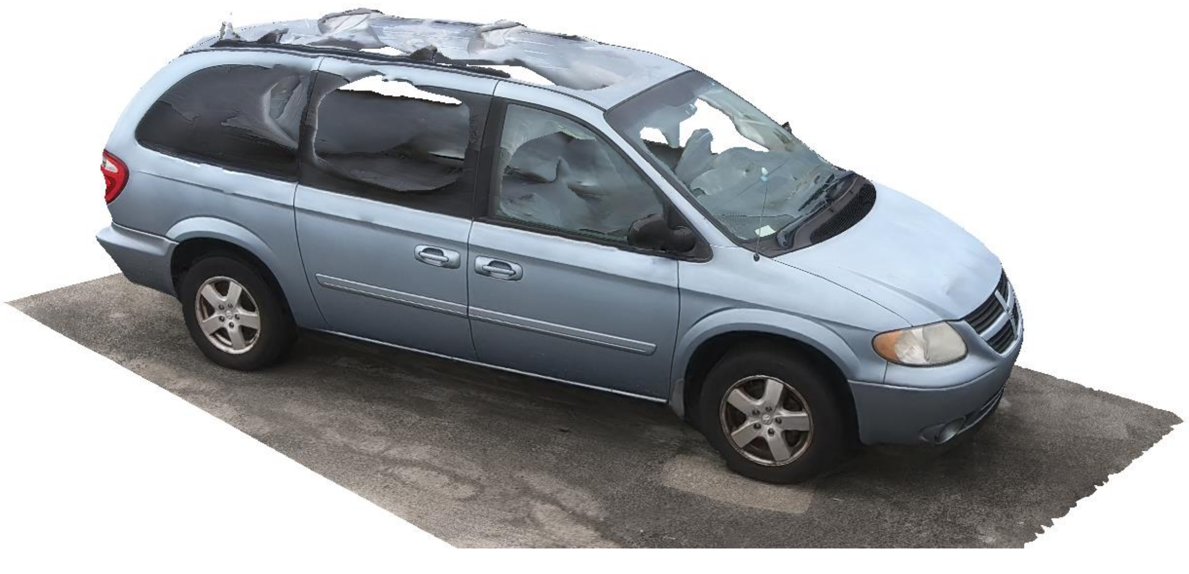


Figure 20: Model recreation errors displayed when using SFM on transparent material.

A test model of a windowed minivan showed significant error where the glass met the metal frame. As such, it is not suggested to use this work on reflective or transparent materials.

3.7 Enclosures and Airframe

The airframe selected for this work is the DJI Inspire Pro 2. While many drones have already been acquired for use in research projects, the Inspire Pro 2 was chosen for best matching the application of SFM model creation. the total takeoff weight is 9.37 lbs and the airframe weight is 7.58 lbs, which means that the photogrammetry module (weighing less than a pound) is well within the total carrying capacity of the DJI. The photogrammetry module is small enough that it can be attached below the Inspire 2 with ground clearance. This allows the Inspire 2 to land without concern of damaging the rover enclosure. The mounting frame for a digital camera used by the Inspire 2 is attached below the UAV, and it is this location that the rover module is attached to. The design of the attachment point on the rover allows for normal operation of the UAV without blocking landing sensors or affecting the landing gear. The antenna used for the U-Blox C099-F9P is mounted above the nose of the UAV. The Inspire 2 is small enough to easily fit in a car for portability, especially when compared to larger drones like the DJI S-1000 or DJI M600 Pro. It requires little effort preparing for data collection flights.

Two enclosures are necessary for the photogrammetry system. The photogrammetry module attached to the drone has a custom fitted housing to attach to the DJI Inspire Pro 2. The attachment point can be modified to accept different models of UAVs if the drone can safely carry the required payload. The camera is carried in a downward facing attachment on the front face with two internal trays housing batteries and the Raspberry Pi 3B and C099-F9P. Small domes are in the lower front corners of the rover module which contain the LED indicators and shield them from excess sunlight which prevents post processing errors during LED state analysis. The model for the rover assembly can be seen in Figure 21.

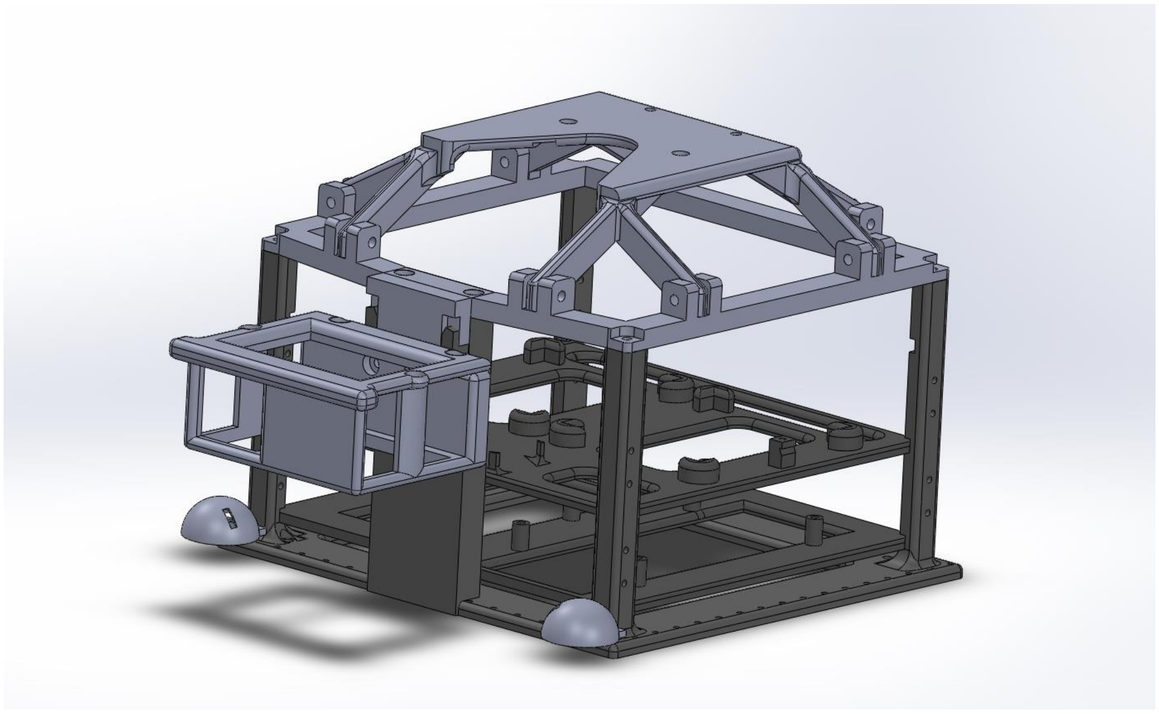


Figure 21: Rover enclosure used to house required hardware for SFM.

The module is printed from Polyethylene terephthalate glycol (PETG) filament with various sizes of machine screws for attachment points. The housing for both enclosures were printed using a Prusa MK3 fused deposition modeling (FDM) three-dimensional printer. The enclosures were modeled using SolidWorks 2018 via a university obtained license. The photogrammetry module installed onto an Inspire 2 Pro can be viewed in Figure 22.

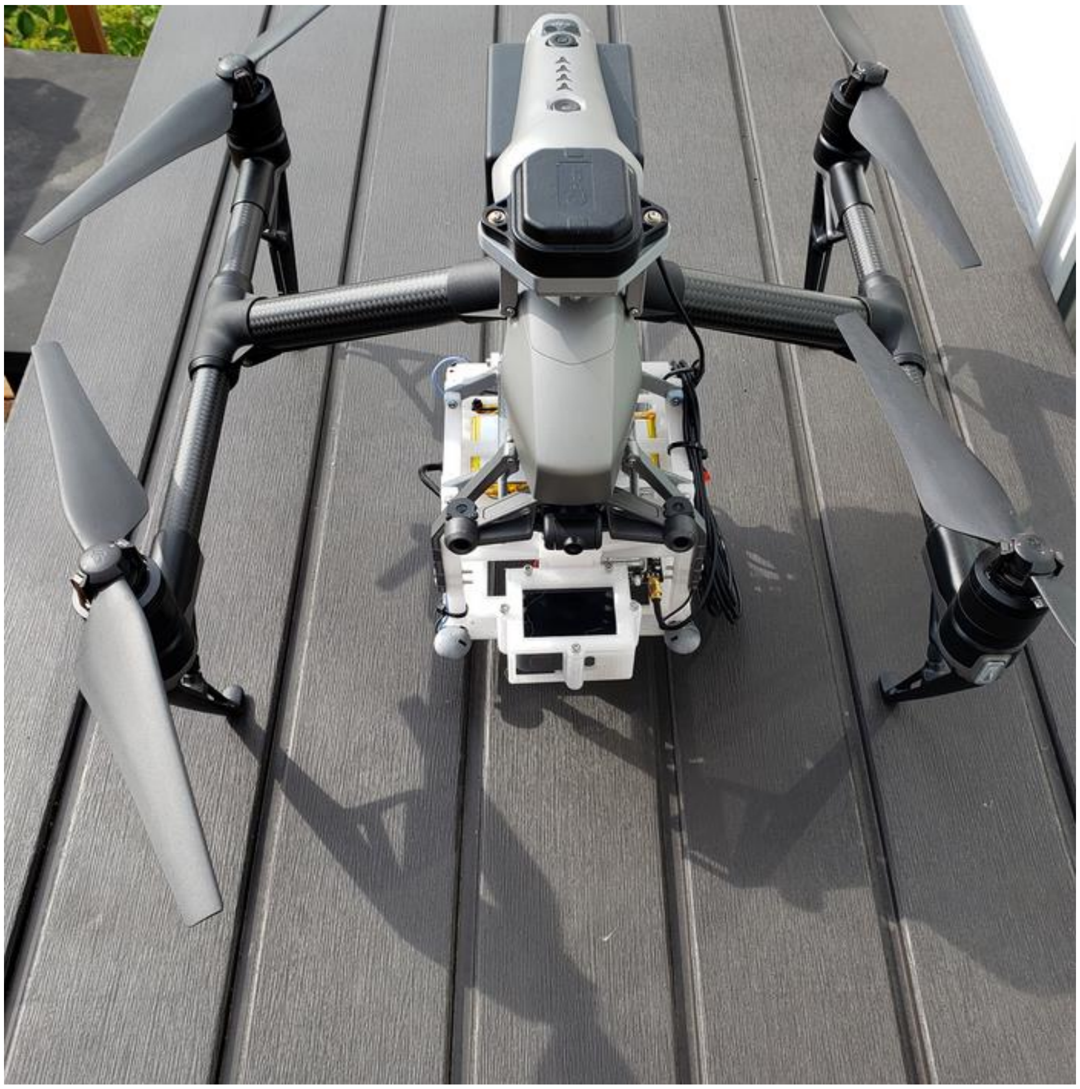


Figure 22: Final design of rover module as attached to DJI Inspire 2 Pro UAV.

The base station module has a similar housing to the rover module as shown in Figure 23.

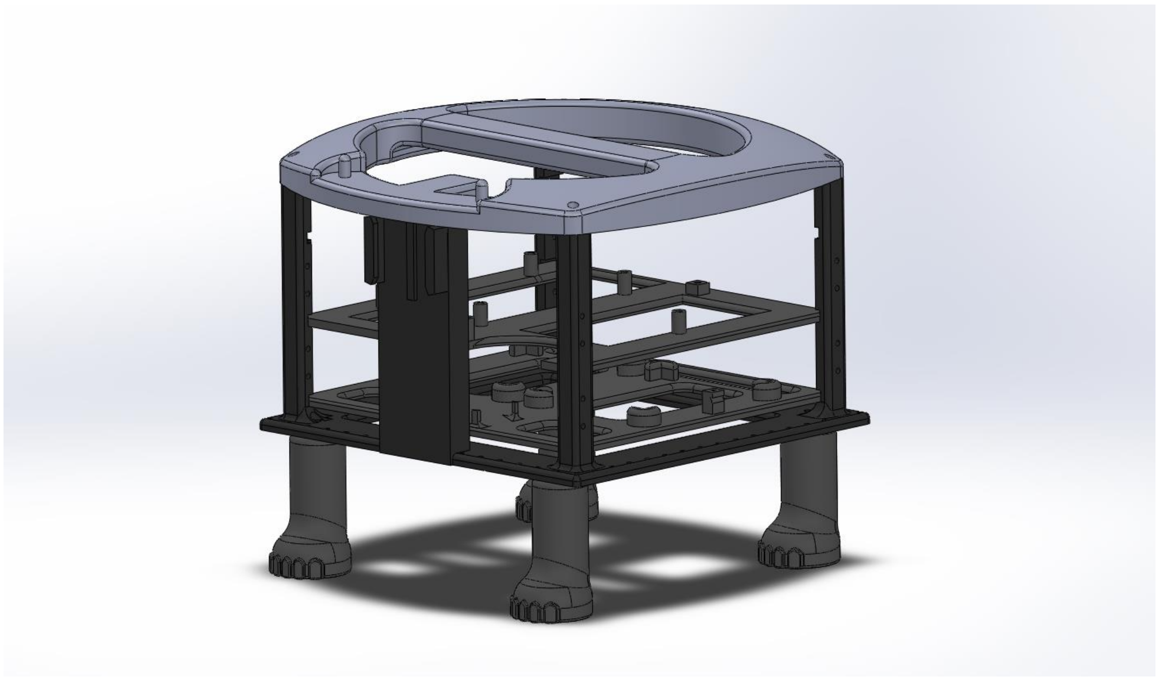


Figure 23: Enclosure used to contain required hardware for base station GCP.

While there is no camera mounting for the base station, there is an adapted holder for the C099-F9P antenna while not in use. During data collection this antenna will be placed on a tripod approximately six feet in elevation for improved reception of positional data. The base station also has four legs to hold the module away from grass or other debris on the ground. A separate tray for batteries and voltage converter is used above a tray holding the Raspberry Pi 3B and U-blox C099-F9P. A small LED holder is attached to the right front edge to display recording status of the base station. The printed base station as used in data collection is displayed in Figure 24.

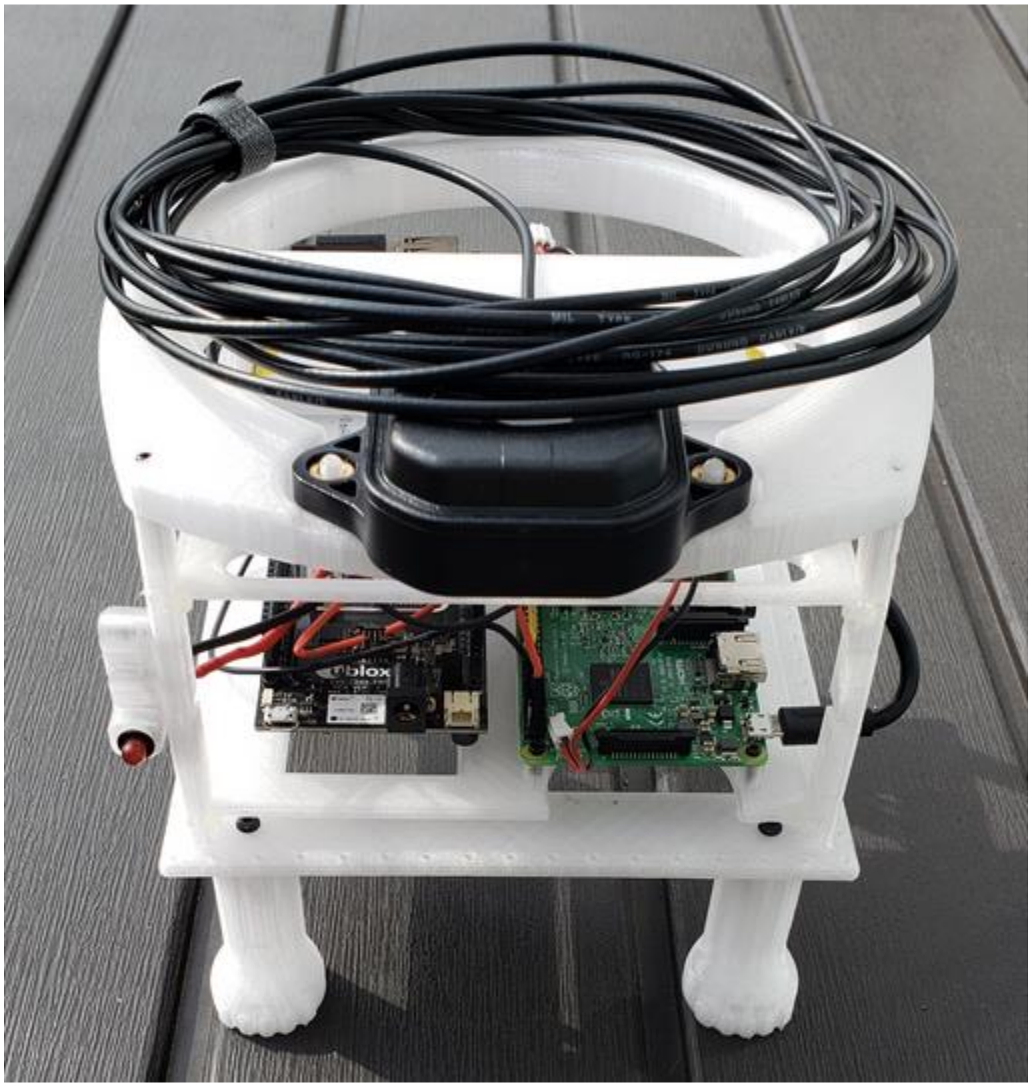


Figure 24: Final design of base station for use as GCP during SFM data collection.

Chapter 4 DATA COLLECTION AND PROCESSING

The goal of the data collection phase is to demonstrate the feasibility of the system as designed in a real-world application. The requirements for a successful data collection flight include:

1) The SFM module must be able to accurately decode and record GNSS data while attached to a UAV in flight.
2) The ground reference must be able to record GNSS data.
3) The camera must be able to capture non-blurry images while moving in flight, and the vibration of the module must remain at a low enough threshold as to not introduce large errors into the SFM model.
4) Finally, there must sufficient GNSS data and images captured for 3D modeling.

The requirements for post processing include:

1) PPK solution must be created.
2) Individual images must be extracted from videos/
3) Time tags for the extracted images must be computed based on LEDs.
4) The PPK solution for each image time tag must be extracted

4.1 Data Collection

In order to verify the requirements for data collection, two canvas targets were constructed with tent-like shapes. Both targets are placed a short distance from one another. They are identified with different colors, one in black and one in blue, and are displayed in Figure 26. The pattern applied to each tent includes a lot of corners, which are used as point features to recreate SFM models. The location, physical dimensions and patterns (corners) of both targets will be carefully measured and used as a truth reference. Based on that, the performance of the point cloud, including accuracy and precision will be evaluated.

The proposed system will also be compared against previously validated technologies. In addition to the proposed SFM system, a UAV-based Light Detection and Ranging (LiDAR) is also used to collect a point cloud. This LiDAR system based on a SICKLD-MRS420201 LiDAR and a DJI Matrice 600 Pro UAV. The LiDAR ranging noise is approximately 0.04 m 1 sigma . The performance of the LiDAR point cloud will be compared against that of SFM. Furthermore, the LiDAR point cloud are geo-referenced, which means it can be used to extract GCPs for the SFM point cloud.

During the flight, the Inspire 2 Pro is flown in circular patterns at various heights above the targets while recording for approximately ten minutes as seen in Figure 27. After landing, the data is checked for integrity and the session is concluded to be followed by the later post processing stages. The base station antenna attached to the top of a tripod, shown in Figure 25. A more in-depth explanation of the data collection process follows. The process for data collection is shown in a flow diagram in Figure 28.

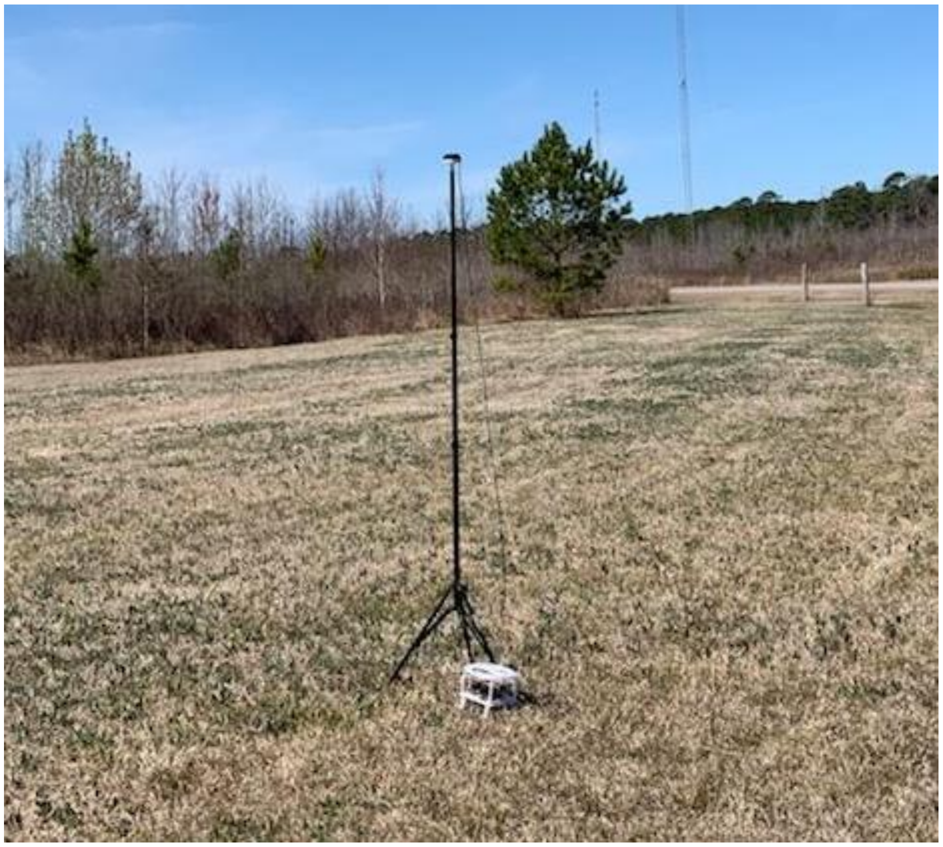


Figure 25: Base station GCP in use with antenna tripod during data collection.

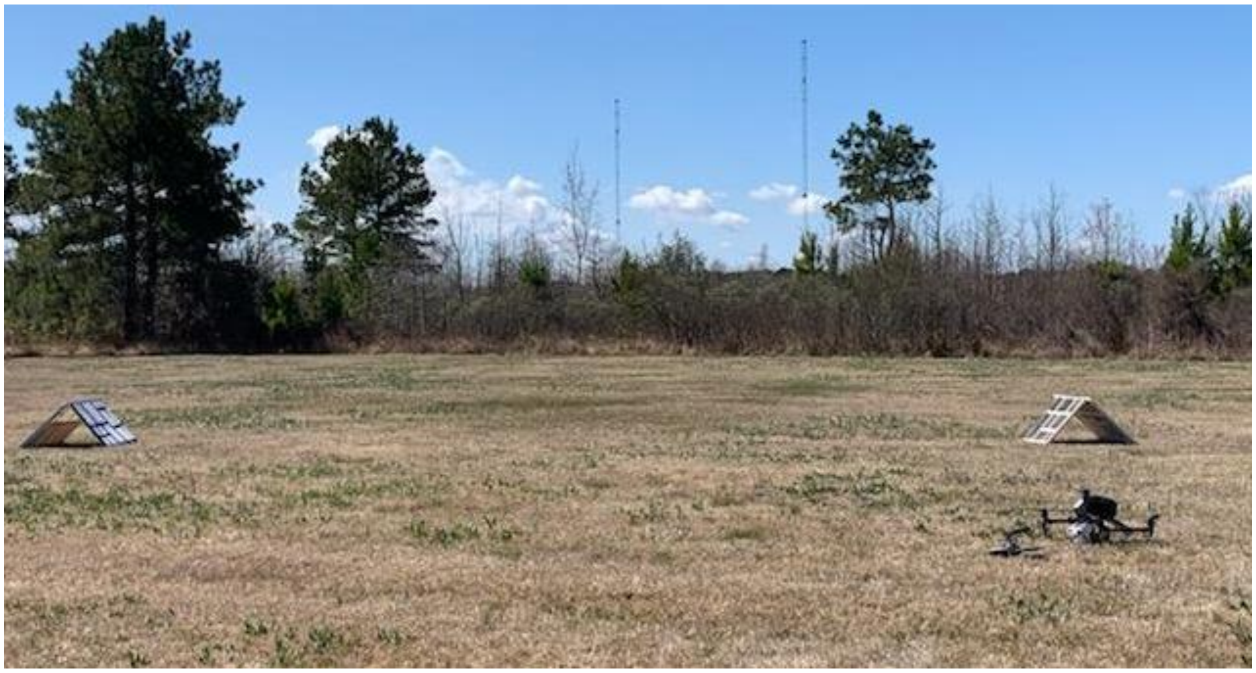


Figure 26: Rover preparing to begin data collection process of featured targets.

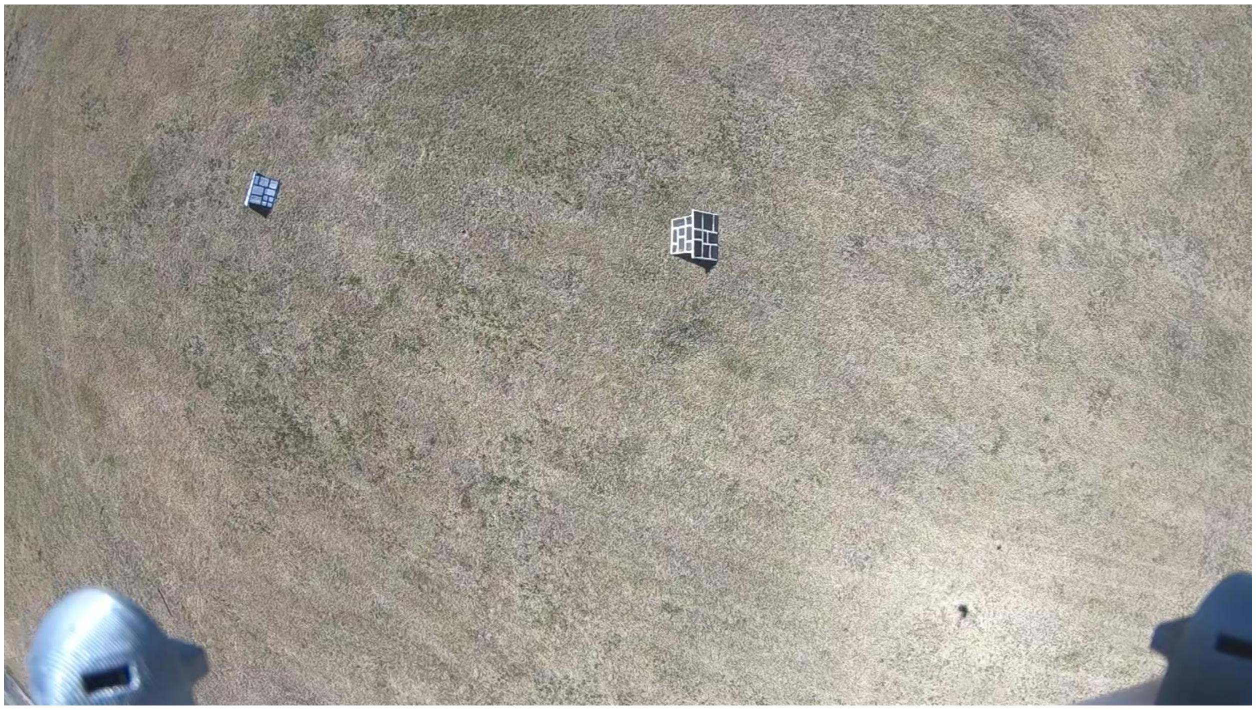


Figure 27: Targets as seen by rover during data collection.

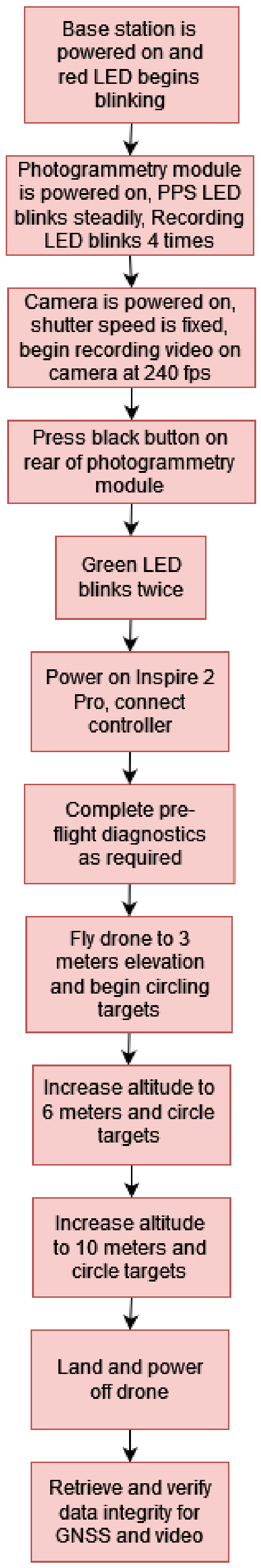

Base station is powered on and red LED begins blinking
Photogrammetry module is powered on, PPS LED blinks steadily, Recording LED blinks 4 times
Camera is powered on, shutter speed is fixed, begin recording video on camera at 240 fps
Press black button on rear of photogrammetry module
Green LED blinks twice
Power on Inspire 2 Pro, connect controller
Complete pre-flight diagnostics as required
Fly drone to 3 meters elevation and begin circling targets
Increase altitude to 6 meters and circle targets
Increase altitude to 10 meters and circle targets
Land and power off drone
Retrieve and verify data integrity for GNSS and video

Figure 28: The method of data collection used to ensure data completeness and integrity.

4.2 Post Processing

As explained in Chapter 3, the images and GNSS data are synchronized prior to model creation. An example of the data found in the reference file created during synchronization is seen in Table 1.

Table 1: Sample positional data related to image in created geo-reference file.

GX010013_001280.jpg	35.63296	-77.4851	-17.6324	0.03	0.03	0.06
GX010013_001520.jpg	35.63296	-77.4851	-17.6614	0.03	0.03	0.06
GX010013_001760.jpg	35.63296	-77.4851	-17.6642	0.03	0.03	0.06
GX010013_001999.jpg	35.63296	-77.4851	-17.663	0.03	0.03	0.06
GX010013_002239.jpg	35.63296	-77.4851	-17.6699	0.03	0.03	0.06
GX010013_002479.jpg	35.63296	-77.4851	-17.6748	0.03	0.03	0.06
GX010013_002719.jpg	35.63296	-77.4851	-17.6651	0.03	0.03	0.06
GX010013_002959.jpg	35.63296	-77.4851	-17.6405	0.03	0.03	0.06
GX010013_003198.jpg	35.63296	-77.4851	-17.6491	0.03	0.03	0.06
GX010013_003438.jpg	35.63296	-77.4851	-17.6503	0.03	0.03	0.06
GX010013_003678.jpg	35.63296	-77.4851	-17.6481	0.03	0.03	0.06
GX010013_003918.jpg	35.63296	-77.4851	-17.653	0.03	0.03	0.06

4.3 Model Creation

The images selected are imported into a separate chunk from the calibration images. The geo-reference file from the same folder is imported after the images. During reference import, the coordinate system and file organization is selected for proper interpretation by Agisoft Metashape. The images (referred to as cameras by the program) are then aligned, then the alignment is optimized.

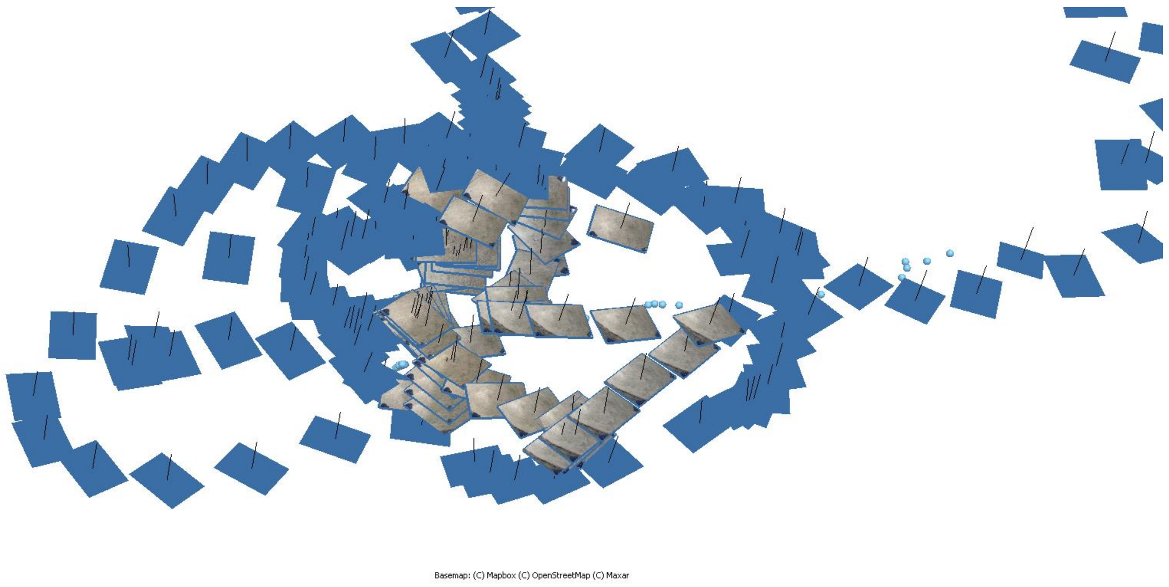


Figure 29: Camera alignment performed by Agisoft Metashape.

Alignment determines the position and orientation of the camera at image capture. This is used to make a sparse point cloud [28].

Figure 30: Sparse point cloud created by Agisoft Metashape.

The dense cloud is then created from the cameras and estimated depth for the cameras. The dense cloud can either be used to build a mesh or exported for further analysis [28].

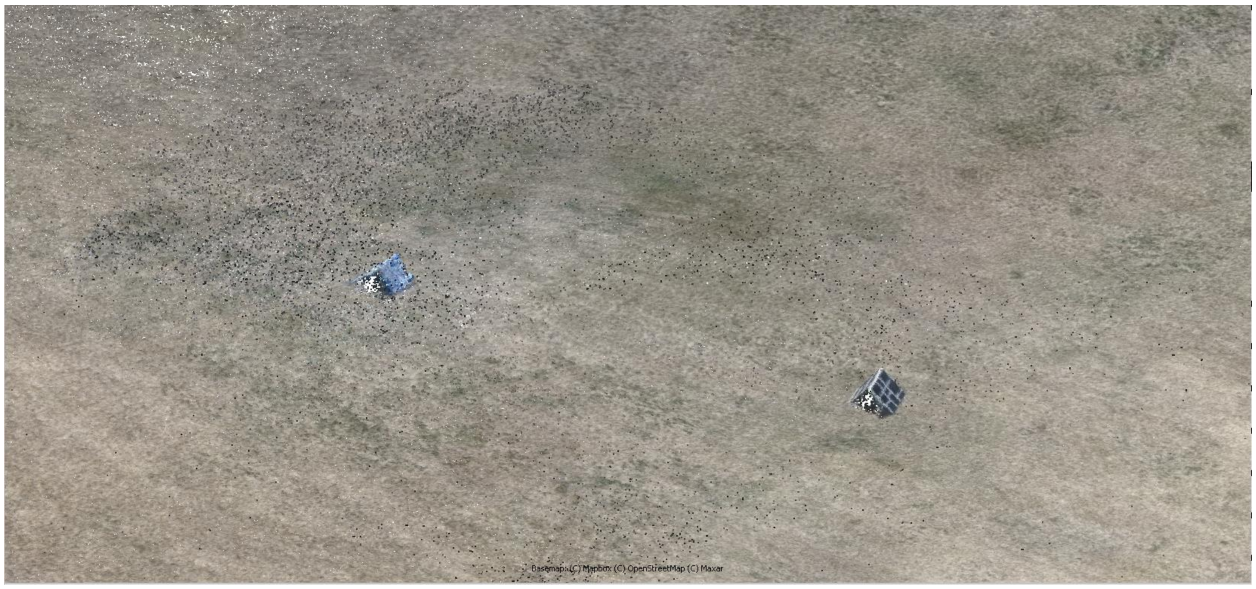


Figure 31: Dense point cloud created by Agisoft Metashape.

The mesh creates the rough model from the point clouds created earlier. A texture can be created and color balanced to properly show an accurate representation of the appearance of the features captured [28].

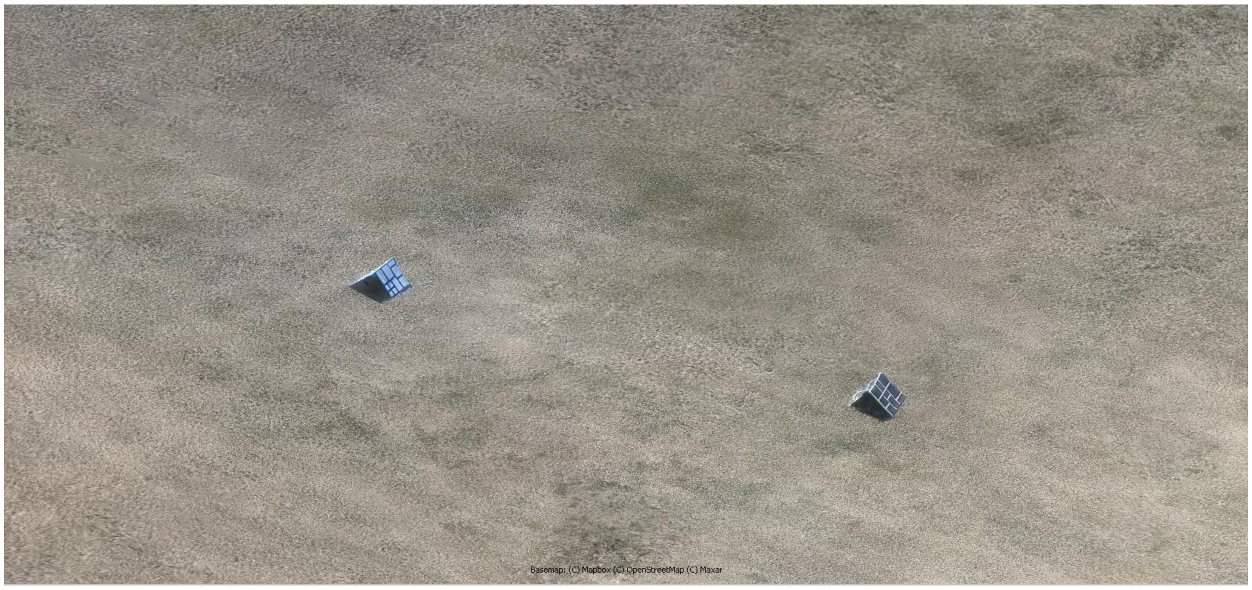


Figure 32: Mesh model created by Agisoft Metashape.

If the model is representing an exceedingly large area then a tile model is suggested for scalable model display. Digital Elevation Model (DEM) can be used to show elevations, cross sections, or various measurements of an uneven surface feature [28]. The targets used for this work are not large enough to require elevation or tiled models. If high resolution images are necessary from the model then an Orthomosaic model can be extracted. Orthomosaic export is a tool used for survey processing of ariel captures or agricultural representations [28].

Chapter 5 RESULTS AND FINDINGS

A three-dimensional model of the scanned area was recreated in Agisoft Metashape using point clouds derived from the images and positional data gathered.

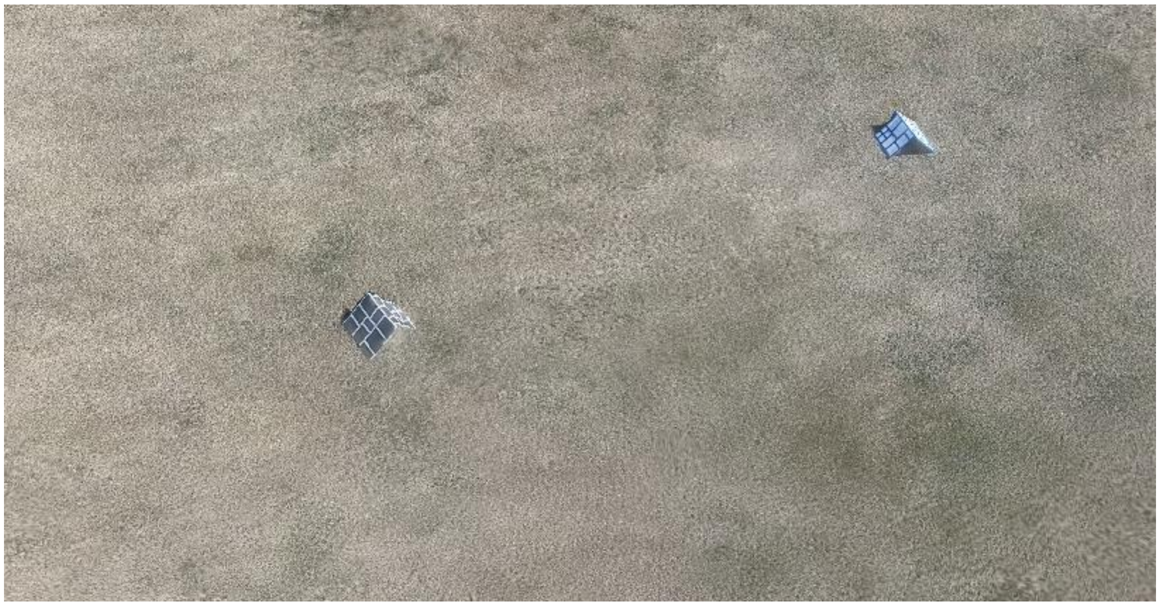


Figure 33: Two targets captured during SFM data collection.

Point features from each individual target were evaluated for precision against manually-measured truth reference.

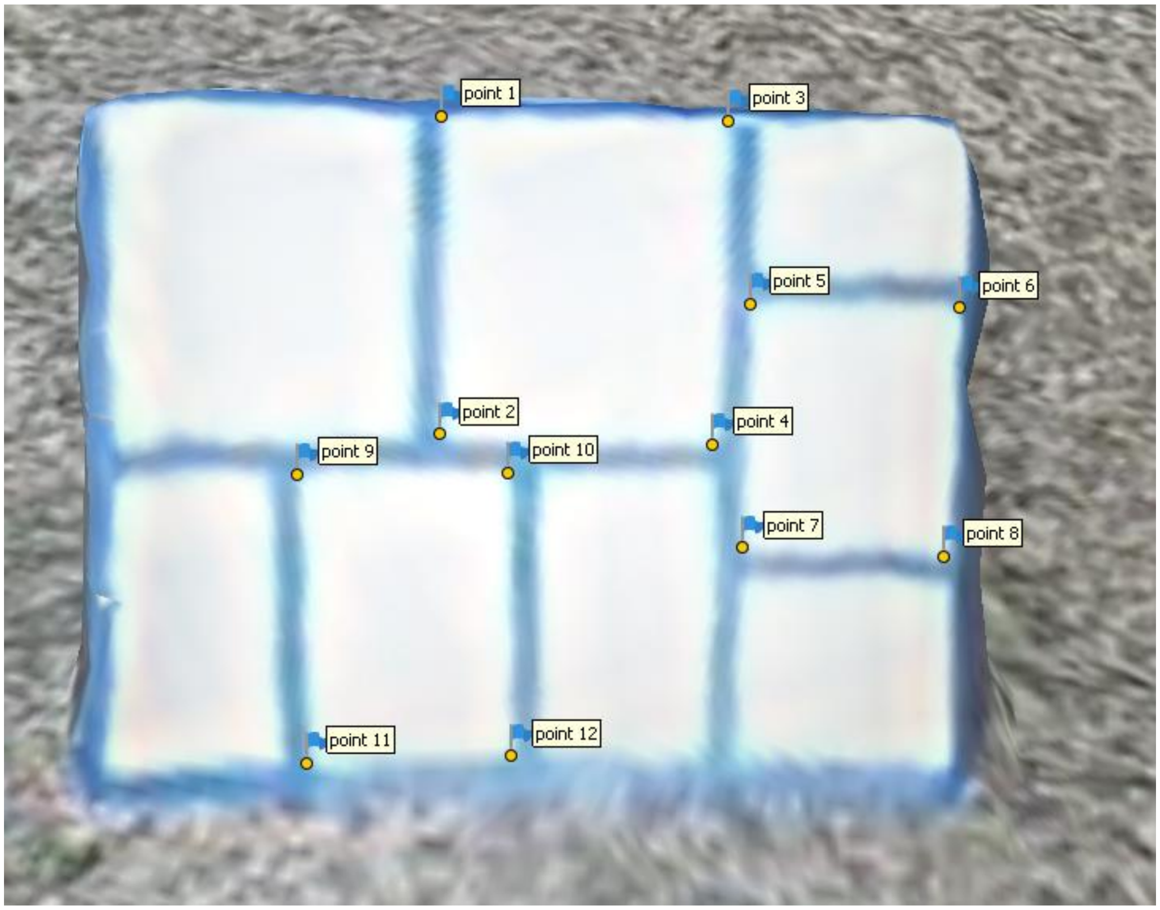


Figure 34: Target one (side A) points selected for measurement analysis.

As shown in Figure 34, points from Target one (side A) were selected where the corners of the shapes were clearly defined and could be clicked accurately within the Agisoft Metashape workspace. The distance between two points was then compared with manual measurements of the actual targets. The average error is 4.99 mm with a standard deviation of 3.97 mm, shown in Table 2.

Table 2: Target one (side A) Measurements

Points	Length (m)	Actual (m)	Error (m)	Error ABS (m)	STDEV
point 1_point 2	0.294498832	0.308	0.013501168	0.013501168	0.003974819
point 1_point 3	0.273345788	0.276	0.002654212	0.002654212	
point 2_point 4	0.262243697	0.26	-0.002243697	0.002243697	
point 3_point 4	0.305712611	0.305	-0.000712611	0.000712611	
point 5_point 6	0.205831182	0.202	-0.003831182	0.003831182	
point 5_point 7	0.231472785	0.233	0.001527215	0.001527215	
point 6_point 8	0.239649694	0.231	-0.008649694	0.008649694	
point 7_point 8	0.195995471	0.201	0.005004529	0.005004529	
point 9_point 10	0.196675302	0.193	-0.003675302	0.003675302	
point 9_point 11	0.271252559	0.275	0.003747441	0.003747441	
point 10_point 12	0.267715995	0.271	0.003284005	0.003284005	
point 11_point 12	0.193084828	0.182	-0.011084828	0.011084828	

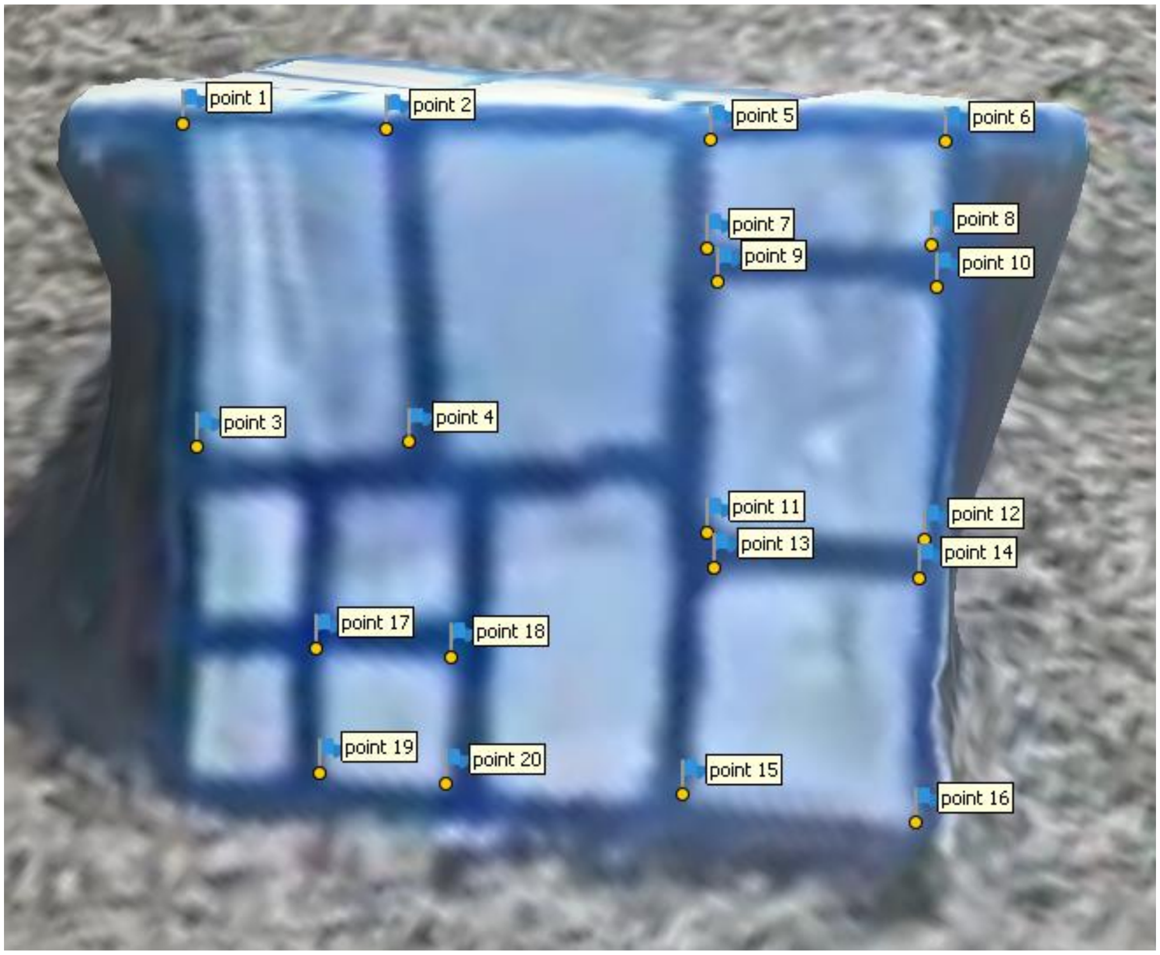


Figure 35: Target one (side B) selected for measurement analysis.

The opposing face (side B) of target one has a similar range of target points taken and examined for accuracy against manual measurements. The average error is 9.09 mm with a standard deviation of 6.46 mm. Measurements may be viewed in Table 3.

Table 3: Target one (side B) Measurements

Points	Length (m)	Length Actual (m)	Error (m)	Error ABS (m)	STDEV
point 1_point 2	0.169739605	0.17	0.000260395	0.000260395	0.006458
point 1_point 3	0.268863691	0.28	0.011136309	0.011136309	
point 2_point 4	0.257462245	0.272	0.014537755	0.014537755	
point 3_point 4	0.181203474	0.176	-0.005203474	0.005203474	
point 5_point 6	0.187313446	0.191	0.003686554	0.003686554	
point 5_point 7	0.08676942	0.088	0.00123058	0.00123058	
point 6_point 8	0.081998202	0.093	0.011001798	0.011001798	
point 7_point 8	0.179538094	0.19	0.010461906	0.010461906	
point 9_point 10	0.175819755	0.191	0.015180245	0.015180245	
point 9_point 11	0.204901009	0.217	0.012098991	0.012098991	
point 10_point 12	0.203093282	0.212	0.008906718	0.008906718	
point 11_point 12	0.17707746	0.193	0.01592254	0.01592254	
point 13_point 14	0.167470813	0.192	0.024529187	0.024529187	
point 13_point 15	0.189149696	0.177	-0.012149696	0.012149696	
point 14_point 16	0.198209845	0.186	-0.012209845	0.012209845	
point 15_point 16	0.192393606	0.19	-0.002393606	0.002393606	
point 17_point 18	0.114852662	0.102	-0.012852662	0.012852662	
point 17_point 19	0.106341186	0.106	-0.000341186	0.000341186	
point 18_point 20	0.106905621	0.107	9.43786E-05	9.43786E-05	
point 19_point 20	0.108622621	0.101	-0.007622621	0.007622621	

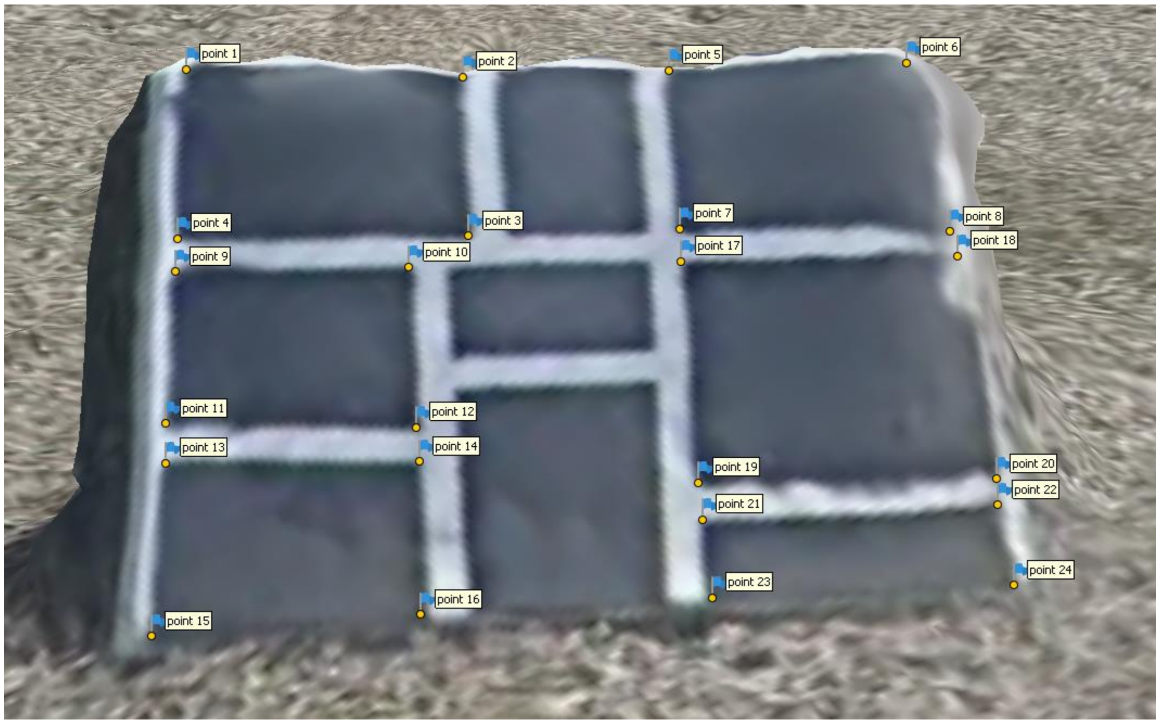


Figure 36: Target two (side A) selected for measurement analysis.

Target two (side A) was analyzed with point measurements compared to manual measurements. The average difference was 15.36 mm with a standard deviation of 7.95 mm, as shown in Table 4. The opposing face of target two had insufficient images for feature representation and was interpolated by Agisoft Metashape. The result was not applicable for point selection.

Table 4: Target two (side A) Measurements

Points	Length (m)	Actual Length (m)	Error (m)	Error ABS (m)	STDEV
point 1_point 2	0.29680038	0.307	0.0102	0.01019962	0.007947
point 1_point 4	0.203354787	0.226	0.022645	0.022645213	
point 2_point 3	0.198398381	0.216	0.017602	0.017601619	
point 3_point 4	0.298862329	0.316	0.017138	0.017137671	
point 5_point 6	0.27939182	0.32	0.040608	0.04060818	
point 5_point 7	0.20291326	0.222	0.019087	0.01908674	
point 6_point 8	0.217788332	0.236	0.018212	0.018211668	
point 7_point 8	0.304943161	0.321	0.016057	0.016056839	
point 9_point 10	0.235596942	0.25	0.014403	0.014403058	
point 9_point 11	0.166172381	0.169	0.002828	0.002827619	
point 10_point 12	0.177141197	0.184	0.006859	0.006858803	
point 11_point 12	0.242337932	0.261	0.018662	0.018662068	
point 13_point 14	0.242753413	0.26	0.017247	0.017246587	
point 13_point 15	0.170768098	0.185	0.014232	0.014231902	
point 14_point 16	0.158823378	0.172	0.013177	0.013176622	
point 15_point 16	0.246765991	0.268	0.021234	0.021234009	
point 17_point 18	0.309568814	0.317	0.007431	0.007431186	
point 17_point 19	0.249699198	0.26	0.010301	0.010300802	
point 18_point 20	0.262518676	0.26	-0.00252	0.002518676	
point 19_point 20	0.311158934	0.325	0.013841	0.013841066	
point 21_point 22	0.306176678	0.327	0.020823	0.020823322	
point 21_point 23	0.082472747	0.09	0.007527	0.007527253	
point 22_point 24	0.088403719	0.101	0.012596	0.012596281	
point 23_point 24	0.302585235	0.326	0.023415	0.023414765	

The dense point cloud created of the targets was also compared to a LiDAR point cloud of the same targets during the data collection event. The point clouds were both imported into MATLAB. The SFM point cloud was rotated manually first, to achieve a rough alignment with LiDAR. After that, it can be registered to the LiDAR point cloud automatically in MATLAB, which results in an accurate rotation of the SFM model in the world frame. The SFM model can now be registered to the LiDAR model, and indirectly to the world frame. Therefore, the LiDAR points are effectively used as GCPs. It can be repeated for both targets individually, and the registration error has been computed in MATLAB.

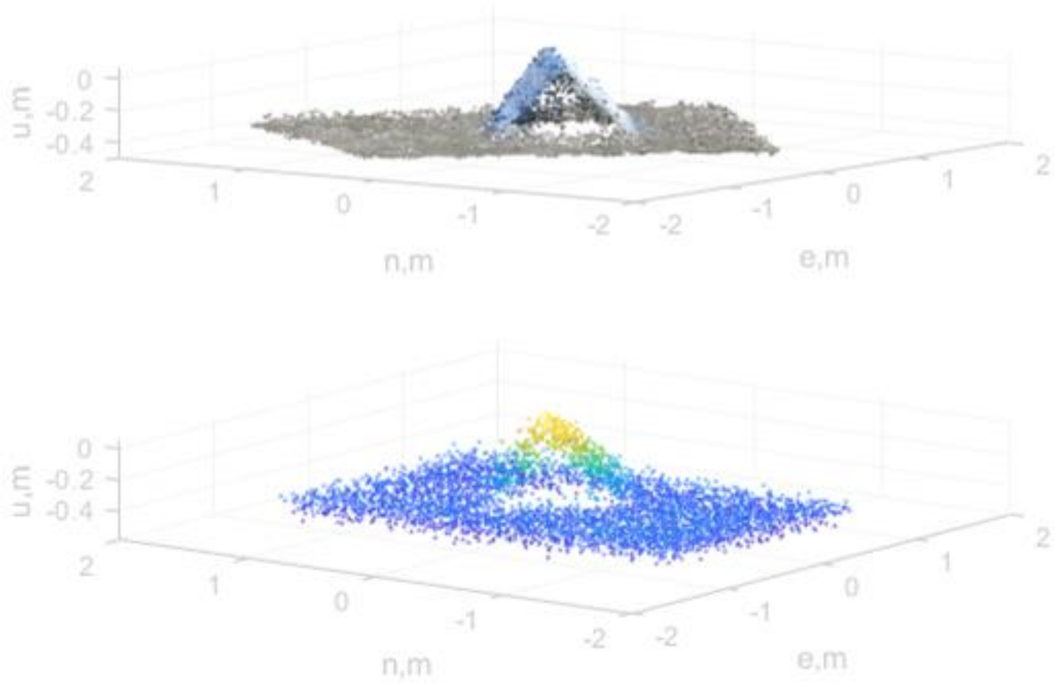


Figure 37: Courtesy of Dr. Zhen Zhu, East Carolina University (2022)

Individually, the geo-registration error for a target has a standard deviation of approximately 0.04 m, which is close to the LiDAR ranging noise level. There is strong agreement between SFM and LiDAR models for each target after geo-registration.

However, if both targets are used simultaneously as GCPs, the comparison shows a drift between both targets in the SFM point cloud. One of the targets will have a bias of approximately 0.1 m in geo-registration.

The distance between point features evaluates the relative precision of the models created in this work. The distance measurements from both targets have milli-meter accuracy, which is adequate for SFM and comparable to other UAV-based SFM technologies. Similar UAV SFM approaches have resolutions of centimeter accuracy [4], error ranges from 16.4 to 23.5 cm for geo-referencing accuracy of large terrain [23], and RMSE of approximately one cm [21]. With a standard deviation of three to eight millimeters, the valid target measurements appear to be at or below the general accuracy found in literature when observing single targets individually.

The resolution of the SFM point cloud is sufficiently high for imagery representation in the 3D model. For example, it shows much higher resolution when compared to LiDAR point clouds as seen in Figure 37.

The definition of absolute accuracy of SFM, however, is more ambiguous. In theory, it can be evaluated by comparing the orientation and location of the same targets in the point cloud from SFM and LiDAR. However, the absolute accuracy of SFM primarily depends on the accuracy of GCPs. As afore mentioned, without known GCPs in the SFM model, geo-registration of the model is still problematic even with PPK. The SFM model has an unknown rotation with respect to the world frame, which makes a direct comparison meaningless.

The addition of few GCPs during data collection could assist in fixing the SFM orientation problem. GCPs can be pre-surveyed targets, or extracted objects from the LiDAR point cloud. Unlike SFM, the LiDAR point cloud has the benefit of orientation and position measurements. When the LiDAR point cloud of each target is used as GCP for SFM, LiDAR and SFM showed a good agreement, with 0.04 m absolute accuracy. It is primarily due to combined relative error from SFM and LiDAR. However, it is interesting to observe that when both targets were used as GCP simultaneously, there appears to be a greater absolute error (0.1 m bias).

It appears that one of the targets may be incorrectly located and/or oriented relative to the other in the SFM model. Since both targets were separately placed on a grass field with a long distance between them, it could indicate that the grass field between them were not correctly modeled in SFM. While the precision of each target is adequate, the absolute accuracy of the SFM could be further improved. For example, the addition of GCPs between the targets may alleviate some of the problem, improving the accuracy of the work.

Chapter 6 CONCLUSIONS AND FUTURE WORK

An open-interface SFM system is based on a small UAV has been presented in this work. In this system, a GoPro Hero 7 camera was used to capture video at 239.75 fps. For a PPK solution, the U-Blox C099-F9P receiver captured and relayed GNSS data to a Raspberry Pi 3B for recording. 3.7-volt Lithium Polymer batteries powered all the components in the system, which is independent of the UAV power. The performance of this system has been successfully demonstrated in a flight test, in which 3D point cloud models were created and validated. The precision of this point cloud is similar to what has been reported in the literature. The errors assessed with two calibration targets have 1 sigma values of 3-8mm.

The main focus of this work is on the open-interfaces for data collection and open-source post processing solutions. The data and synchronization interfaces between the camera and GNSS are open and interchangeable, which means either component could be independently replaced or upgraded. An open-source GNSS post processing library was used, which can handle GNSS data in a large variety of formats. For the sake of convenience, Agisoft Metashape is used create 3D point cloud models in this work. Although it is not open-source software, it can be easily replaced by open-source options in the future.

The performance was made possible by an open-interface timing mechanism, which is one of the original contributions of this work. The images captured by the video camera are properly synchronized to GNSS data via a LED-based timing circuitry and open-source post processing software. With proper timing, the proposed low-cost system functions as well as the proprietary solutions.

A system for gathering images and positional data in order to build a three-dimensional model is described in this work. The position of the drone is captured in flight and used to determine the position of the camera. The images captured are paired with positional data, and from these pairings, a three-dimensional model is created. The process is completed with individual hardware components, available off-the-shelf and for a low cost. The preparation for the modeling program is done with open-source programs, which are freely made available. The models created are recreations of the original object within a centimeter, assuming only one object is recreated at a time. The outcome is a readily available, cheaper option for creating three-dimensional models of real-world objects.

Below are comments and recommendations regarding each subsystem.

6.1 Data Collection

While the models created were enough for an initial analysis of the validity of the photogrammetry results, there are still areas to be improved upon. The area being captured by the Hero 7 was difficult to estimate while the drone was in flight. Therefore, instead of a circling pattern when capturing target data, it is suggested that a grid like pattern is adopted instead. This would reduce errors like those seen on the second side of target two in Figure 38.

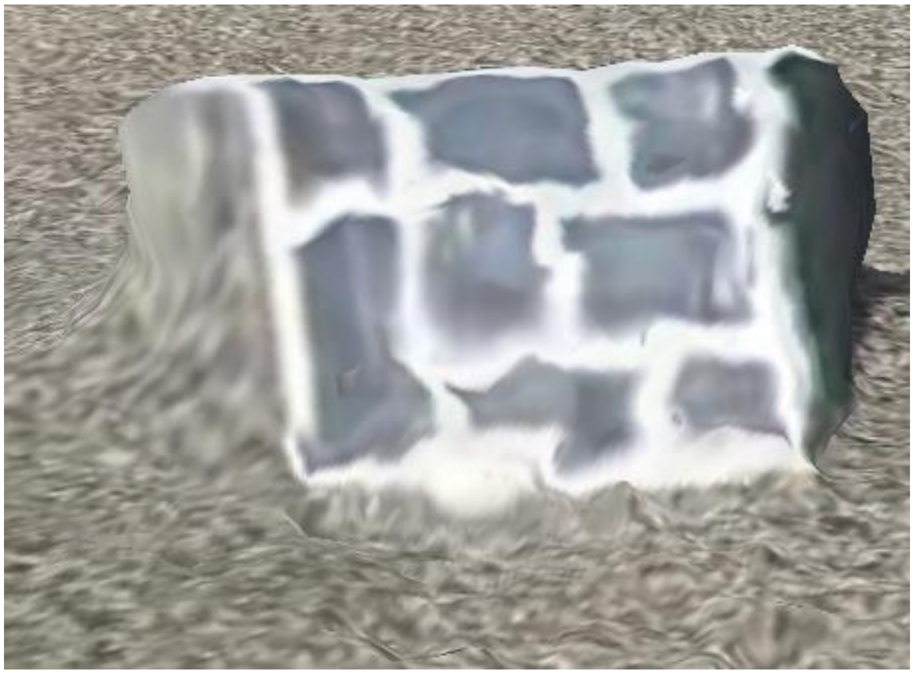


Figure 38: Target two (side B) was invalid due to insufficient image coverage.

Use of a circling pattern caused an oversight in area coverage, requiring Agisoft Metashape to instead attempt to interpolate the pattern present on target two. This attempt invalidated the results and caused significant error to the modeling of target two. A grid like pattern may help to alleviate the modeling issue caused by insufficient coverage of the targets if used in a slowly incremental pattern.

When copying the positional data from the rover and base station it is recommended to use the copy function instead of the cut function. Data loss was observed when using the cutting function to migrate positional data from the Micro-SD cards to the laptop resulting in lost time while data collection was repeated. By copying the data first, there is a file size present that can be observed from the original file which can be used to verify data integrity of the copy. The original file may then be deleted after verification is complete.

Geo-registered SFM models has higher resolution and precision than LiDAR, although it needs help from GCPs or LiDAR. A combination of the two solutions could be best for both an increase in accuracy and a method of validation without requiring manual measurements of real-world objects.

For LiDAR comparison it is recommended to also capture the GCPs used for both the SFM process and the LiDAR scan as well. This will allow for easier alignment of the models by giving anchor points by which to align the models. Without the LiDAR GCPs present in the photogrammetry model, alignment becomes significantly more time consuming.

The three-dimensional models created had results of properly surveyed targets with accuracy averaging at less than a centimeter error. Compared to the accuracy seen in current products and other methods seen in the literature, the photogrammetry system designed functions at an acceptable level of performance for single targets. Biasing errors are present when comparing multiple targets at the same time. The bias error is easiest to observe when comparing the model created with SFM methods to the models created at the same event using LiDAR technology.

While the SFM used in this model is precise, it can only be stated that it is sufficient when concerned with a single target at a time. If multiple targets were scanned with no GCP in between them, one of the targets is slightly off position. A standard deviation of 5 cm was seen in the position of the second observed target if compared to the position of the first target in the same model. Regardless of which target the LiDAR model was calibrated to the SFM model, the secondary target was slightly out of orientation. The cause may be a scaling error from the SFM process. Investigations using GCPs placed between multiple targets are recommended to determine if the error is originating from the SFM process.

6.2 Enclosure

The enclosure for the rover and base station in the photogrammetry system showed that 3D SolidWorks modeling and printing are excellent tools for designing and testing new systems. While multiple versions of the final product were created and refined, the ability to print models rapidly and accurately on site without the need to wait for parts to be shipped simplified the design process immeasurably. Material limitations were present, however. The models used were printed in PETG for the slight flexibility and light weight material. This material worked well for initial versions but for a more permanent installation onto a drone it is suggested to use a slightly stiffer material such as those outlined in the future work section. PETG rover casings required a slight redesign to account for vibrations shaking the rover module in flight and moving the camera in relation to the C099-F9P receiver. This may contribute to errors seen in the final models. Stiffer materials will not be required for the base station as it is not subjected to the same level of vibrations as the rover module.

Single connector wire harnesses are not recommended in future work. Vibrations during transport and during flight cause intermittent connectivity which results in potential positional data loss. While LED indicators (indicating message traffic) mitigate this issue, it is instead suggested that single headers with multiple pin inputs be used instead for a more secure connection.

The enclosure features a cage for the Hero 7 which uses screws to ensure it remains closed during flight. A latching system may simplify data retrieval after flight by removing the need to manipulate screws requiring specific tools in the field.

The rover enclosure suffered from shaking during data collection flights. Stiffer material should be used, perhaps carbon reinforced nylon. There was significant shaking of the camera in the enclosure during windy conditions. This caused an oscillation in the measurement between the camera and C099-F9P antenna and introduced error into the final model. Temporary measures for the PETG based enclosure were taken by adding reinforcements from the cage to the drone, requiring more attachment points being used on the drone and further complicating module removal.

It is the hope of this author that this work will be used to further refine open-source methods of data collection for modeling purposes. It is with gratitude to Dr. Zhen Zhu for his guidance and contributions that this body of work was created and the current level of open-source solutions for modeling real world features is furthered.

References

[1] M. J. Westoby, J. Brasington, N. F. Glasser, M. J. Hambrey and J. M. Reynolds, "'Structure-From-Motion' photogrammetry: A Low Cost, Effective Tool for Geoscience Applications," *Geomorphology,* vol. 179, pp. 300-314, 2012.

[2] X. Zhuo, T. Koch, F. Kurz, F. Fraundorfer and P. Reinartz, "Automatic UAV Image Geo-Registration by Matching UAV Images to Georeferenced Image Data," *Remote Sensing,* vol. 9, no. 376, 2017.

[3] J. E. Hammond, C. A. Vernon, T. J. Okeson, B. J. Barrett, S. Arce, V. Newell, J. Janson, K. W. Franke and J. D. Hedengren, "Survey of 9 UAV Set-Covering Algorithims for TErrain Photogrammetry," *Remote Sensing,* vol. 12, no. 14, 2020.

[4] S. Sledz, M. Ewertowski and J. Piekarczyk, "Applications of unmanned aerial vehicle (UAV) surveys and Structure from Motion Photogrammetry in Glacial and Periglacial Geomorphology," *Geomorphology,* vol. 378, no. 107620, 2021.

[5] P. E. Carbonneau and J. T. Dietrich, "Cost Effective non-metric photogrammetry from Consumer-Grade sUAS: Implications for direct georeferencing of structure from motion photogrammetry.," *Earth Surface Processes and Landforms,* vol. 65, pp. 1722-1729, 2020.

[6] N. Cody, "Flight and Federalism: Federal Preemption of State and Local Drone Laws," *Washington Law Review,* vol. 93, no. 3, pp. 1495-1531, 2018.

[7] H. Wang, Y. Duan, Y. Shi, Y. Kato, S. Ninomiya and et al., "EasyIDP: A Python Package for Intermediate Data Processing in UAV-Based Plant Phenotyping," *Remote Sensing,* vol. 13, no. 13, 2021.

[8] R. R. Cunha, C. T. Arrabal, M. M. Dantas and H. R. Bassaneli, "Laser scanner and drone photogrammetry: A statistical comparison between 3-dimensional models and its impacts on outdoor crime scene registration," *Forensic Science International,* vol. 330, 2022.

[9] L. Froideval, K. Pedoja, F. Garestier, P. Moulon, C. Conessa, X. Pellerin Le Bas, K. Traore and L. Benoit, "A low-cost open-source workflow to generate georeferenced 3D SfM photogrammetric models of rocky outcrops," *Photogram Rec,* vol. 34, no. 168, pp. 365-384, 2019.

[10] M. E. Hodgson, "On the Accuracy of Low-Cost Dual-Frequency GNSS Network Receivers and Reference Data," *GI Science and Remote Sensing,* vol. 57, no. 7, pp. 907-923, 2020.

[11] D. T. I. Bayley and A. O. M. Mogg, "A Protocol for the Large-Scale Analysis of Reefs using Structure from Motion Photogrammetry," *Methods in Ecology and Evolution,* vol. 11, pp. 1410-1420, 2020.

[12] O. E. Mora, J. Chen, P. Stoiber, Z. Koppanyi, D. Pluta, R. Josenhans and M. Okubo, "Accuracy of Stockpile Estimates Using Low-Cost sUAS Photogrammetry," *International Journal of Remote Sensing,* vol. 41, no. 12, pp. 4512-4529, 2020.

[13] S. Guan, Z. Zhen and G. Wang, "A Review on UAV-Based Remote Sensing Technologies for Construction and Civil Applications," *Drones,* vol. 6, no. 5, 2022.

[14] H. J. Larsen and M. R. Bennet, "Empirical Evaluation of the Reliability of Photogrammetry Software in the Recovery of Three-Dimensional Footwear Impressions," *Journal of Forensic Sciences,* vol. 65, no. 5, pp. 1722-1729, 2020.

[15] C. Nicolae, E. Nocerino, F. Menna and F. Remondino, "Photogrammetry Applied to Problematic Artefacts," *The International Archives of the Photogrammetry, Remote Sensing, and Spatial Information Sciences,* vol. XL, no. 5, pp. 451-456, 2014.

[16] M. Hobart, M. Pflanz, C. Weltzien and M. Schirrmann, "Growth Height Determination of Tree Walls for Precise Monitoring in Apple Fruit Production Using UAV Photogrammetry," *Remote Sensing,* vol. 12, no. 10, 2020.

[17] M. K. Zaslavskiy, "Review of Photogrammetry Techniques for 3D scanning Tasks of Buildings," in *Proceedings of the 28th Conference of Open Innovations Association FRUCT*, 2021.

[18] E. F. Berra, E. F. Berra and M. V. Peppa, "Advances and Challenges of UAV SFM MVS Photogrammetry and Remote Sensing: Short Review," *The International Archives of the Photogrammetry, Remote Sensing, and Spatial Information Sciences,* vol. XLII, no. 3, pp. 267-272, 2020.

[19] A. Federman, M. S. Quintero, S. Kretz, J. Gregg, M. Lengies, C. Ouimet and J. Laliberte, "UAV Photogrammetric Workflows: A Best Practice Guideline," *The International Archives of The Photogrammetry, Remote Sensing and Spatial Information Sciences,* vol. XLII, no. 2, pp. 237-244, 2017.

[20] F. Chiabrando, A. Lingua, P. Maschio and L. L. Teppati, "The Influence of Flight Planning and Camera Orientation In UAVs Photogrammetry. A Test in the Area of Rocca San Silvestro (LI), Tuscany," *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Services,* vol. XLII, no. 2, pp. 163-170, 2017.

[21] P. Gabrlik, A. la Cour-Harbo, P. Kalvodova, L. Zalud and P. Janata, "Calibtration and Accuracy Assessment in a Direct Georeferencing System for UAS Photogrammetry," *International Journal of Remote Sensing,* vol. 39, no. 15, pp. 4391-4959, 2018.

[22] B. Grayson, N. T. Penna and J. P. Mills, "GPS Precise Point Positioning for UAV Photogrammetry," *The Photogrammetric Record,* vol. 33, no. 164, pp. 427-447, 2018.

[23] D. Ekaso, F. Nex and N. Kyle, "Accuracy Assessment of Real-Time Kinematics (RTK) measurements on unmanned aerial vehicles (UAV) for direct geo-referencing," *Geo-Spatial Information Science,* vol. 23, no. 2, pp. 165-181, 2020.

[24] F. Zimmermann, C. Eling, L. Klingbeil and H. Kuhlmann, "Precise Positioning of UAVs - Dealing with Challenging RTK-GPS Measurement Conditions During Automated UAV Flights," *ISPRS Annals of*

the Photogrammetry, Remote Sensing, and Spatial Information Sciences, vol. 4, no. 2, pp. 95-102, 2017.

[25] H. Fazeli, F. Samadzadegan and F. Dadrasjavan, "Evaluating the Potential of RTK-UAV for Automatic Point Cloud Generation in 3D Rapid Mapping," *The International Archives of the Photogrammetry, Remote Sensing, and Spatial Information Sciences,* vol. XLI, no. B6, pp. 221-226, 2016.

[26] A. Ferreira, B. Matias and J. Almeida, "Real-Time GNSS Precise Positioning: RTKLIB for ROS," *International Journal of Advanced Robotic Systems,* vol. 10, 2020.

[27] M. W. Bunder, K. P. Tognetti and G. E. Wheeler, "On binary reflected Gray codes and functions," *Discrete Mathmatics,* vol. 308, no. 9, pp. 1690-1700, 2008.

[28] Agisoft, "Agisoft.com," Agisoft, 2 June 2021. [Online]. Available: https://www.agisoft.com/buy/licensing-options/. [Accessed 2 June, 2021 June 2021].

[29] M. von Ubel, "All3DP.com," February 2021. [Online]. Available: https://all3dp.com/1/best-photogrammetry-software/. [Accessed 2 June 2021].

[30] H. Smith, *What to Look for in Your Photogrammetry Drone,* Copterz.com, 2016.

[31] A. Pinte, "Logiciels.ign.fr," National Institute of Geographic Information and Forestry, 16 March 2017. [Online]. Available: http://logiciels.ign.fr/?Micmac. [Accessed 2 June 2021].

[32] P. Marian, "electroschematics.com," AspenCore, 2021. [Online]. Available: https://www.electroschematics.com/arduino-mega-2560-pinout/. [Accessed 2 June 2021].

[33] A. M. Manual. [Online].

[34] D. Fawcett, B. Azlan, T. C. Hill, L. K. Kho, J. Bennie and K. Anderson, "Unmanned Aerial vehicle (UAV) Derived Structure-From-Motion Photogrammetry Point Clouds for Oil Palm (Elaeis Guineensis) Canopy Segmentation and Height Estimation," *International Journal of Remote Sensing,* vol. 40, no. 19, pp. 7538-7560, 2019.

[35] C. Achille, A. Adami, S. Chiraini, S. Cremonesi, F. Fassi, L. Fregonese and L. Taffurelli, "UAV-Based Photogrammetry and Integrated Technologies for Architectural Applications - Methodological Strategies for the After-Quake Survey of Vertical Structures in Mantua (Italy)," *Sensors,* vol. 15, pp. 15520-15539, 2015.

[36] ECU Division of Research, Economic Development and Engagement, *Unmanned Aircraft Systems (UAS) Regulation - Interim,* Greenville: East Carolina University, 2020.

[37] U-blox, "u-blox.com," U-blox, 2020. [Online]. Available: https://www.u-blox.com/en/product/c099-f9p-application-board. [Accessed 2 June 2021].

[38] UAV Coach, "UAVCoach.com," UAV Coach, 2021. [Online]. Available: https://uavcoach.com/drone-laws-north-carolina/. [Accessed 12 April 2021].

[39] Federal Aviation Administration, *Title 14, Chapter 1, Subchapter F, Part 107 - Small Unmanned Aircraft Systems,* Federal Aviation Administration, 2021.

[40] Pix4D SA, "pix4d.com," Pix4D SA, 2021. [Online]. Available: https://www.pix4d.com/product/pix4dmapper-photogrammetry-software. [Accessed 2 June 2021].

[41] North Carolina Department of Transportation, "NCDOT.com," North Carolina Department of Transportation, 2021. [Online]. Available: https://www.ncdot.gov/news/rr_image_catalog/forms/all%20items%20flat%20view.aspx?rootfolder=/news/rr_image_catalog/logos. [Accessed 2 June 2021].

[42] BangGood, "BangGood.com," [Online]. Available: https://www.banggood.com/Hawkeye-Firefly-Split-4K-160-Degree-HD-Recording-DVR-Mini-FPV-Camera-WDR-Single-Board-Built-in-Mic-Low-Latency-TV-Output-for-RC-Drone-Airplane-p-1528048.htm. [Accessed 2 June 2021].

[43] Adafruit, "Adafruit.com," [Online]. Available: https://www.adafruit.com/product/3055. [Accessed 2 June 2021].

[44] "Sick.com," SICK AG, 2022. [Online]. Available: https://www.sick.com/us/en/detection-and-ranging-solutions/3d-lidar-sensors/ld-mrs/ld-mrs420201/p/p496644?ff_data=JmZmX2lkPXA0OTY2NDQmZmZfbWFzdGVySWQ9cDQ5NjY0NCZmZl90aXRsZT1MRC1NUlM0MjAyMDEmZmZfcXVlcnk9JmZmX3Bvcz0yJmZmX29yaWdQb3M9MiZmZl9wYWdlPTEmZmZfcGFnZ. [Accessed 15 June 2022].

APPENDIX A: ROVER RASPBERRY PI 3B RECORDING AND LED CONTROL

```
import serial
import datetime
import RPi.GPIO as GPIO
import time
import os
GPIO.setmode(GPIO.BCM)
ButtonPin=17
HighForButton=6
LED=26
GPIO.setup(LED, GPIO.OUT)
GPIO.output(LED,GPIO.LOW)
GPIO.setup(ButtonPin, GPIO.IN)
#FireCamera=13
#GPIO.setup(FireCamera, GPIO.OUT)

LastReading=0
StartRecording = 0
StartTime=-55
InBetweenMessagePacketsCounter=0
WaitForEnd=0
NeedHeader=1
RecordingData=0

GPIO.output(LED, 1)
time.sleep(1)
GPIO.output(LED, 0)
time.sleep(1)
GPIO.output(LED, 1)
time.sleep(1)
GPIO.output(LED, 0)
time.sleep(1)
GPIO.output(LED, 1)
time.sleep(1)
GPIO.output(LED, 0)
time.sleep(1)
GPIO.output(LED, 1)
time.sleep(1)
GPIO.output(LED, 0)
time.sleep(1)

savepath='/home/pi/GPSLogs'
filedated = datetime.datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
binfile = os.path.join(savepath, filedated)
binfile += '.bin'
logfile = os.path.join(savepath, filedated)
logfile += '.txt'
startfile='Rover.txt'
startfile=os.path.join(savepath, startfile)
file1=open(logfile,"w")
file2=open(binfile,"w")
file3=open(startfile,"w")
print('Files created')
ser = serial.Serial('/dev/serial0', 115200, timeout = 2,
parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE,
bytesize=serial.EIGHTBITS)
```

```
print("USB Connection Made")
ser.flush()
print("FLUSHED")
FetchStart=0

while StartRecording==0:

      if (GPIO.input(ButtonPin) and (LastReading==0)):
            print('Button Pressed')
            GPIO.output(LED, 1)
            #GPIO.output(FireCamera,1)
            StartRecording=1
            time.sleep(.1)
            GPIO.output(LED,0)

      else:
            print('Not Pressed')
            GPIO.output(LED, 0)
      LastReading=GPIO.input(ButtonPin)
      line=ser.readline()
      file1.write(line)
      file2.write(line)
#sync start times



while FetchStart==0:
#     if ser.in_waiting > 0:
      line = ser.readline()#ser.readline()#ser.readline().decode('utf-
8').rstrip()
      file1.write(line)
      file2.write(line)
      #if(line.find('GNGLL')):
      #     print(line)
      #print(line)
      location=line.find('GNGLL')
      if(location>=0):
            print(line)
            StartTime= int(line[(location+33):(location+39)])
            StartTime=(StartTime+1)
            if(StartTime>=1):


                  FetchStart=1
            #print(StartTime

            #print(type(StartTime))
while WaitForEnd ==0:
      line=ser.readline()
      GPIO.output(LED,1)
      #file3.write("LED ON")
      #file3.write(line)
      location=line.find('GNGLL')
      if(NeedHeader==1):
            StartTimestr=str(StartTime)

            file3.write("*** START TIME " + StartTimestr +" *** \n")
            file1.write("*** START TIME " + StartTimestr +" *** \n")

            NeedHeader=0
      file1.write(line)
      file2.write(line)
      if(location>=0):
            print(line)
            GPIO.output(LED, 0)
            #file3.write(line)
            #file3.write("LED OFF")
            file3.close
            WaitForEnd=1
print(StartTime)
print('Recording data stream to files')
counterforclose=0
while WaitForEnd==1:
      line=ser.readline()
      file1.write(line)
      file2.write(line)
      counterforclose=counterforclose+1
      if counterforclose>10:

            file1.close()
            file2.close()
            file1=open(logfile,"a")
            file2=open(binfile,"a")
            counterforclose=
```

APPENDIX B: BASE STATION RASPBERRY PI 3B RECORDING

```
import serial
import datetime
import os
import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
LED=26
LEDstatus=0
GPIO.setup(LED,GPIO.OUT)
GPIO.output(LED,GPIO.LOW)
print("check serial connection")
if __name__ == '__main__':
      print("found main")
      filedated = datetime.datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
      savepath='/home/pi/GPSLogs'
      binfile = os.path.join(savepath,filedated)
      binfile += '.bin'
      logfile = os.path.join(savepath,filedated)
      logfile += '.log'
      ser = serial.Serial('/dev/serial0', 115200, timeout = 1,
parity=serial.PARITY_NONE, stopbits=serial.STOPBITS_ONE,
bytesize=serial.EIGHTBITS)
      print("USB Connection Made")
      ser.flush()
      print("FLUSHED")
      file1=open(logfile,"w")
      file2=open(binfile,"w")
      counterforclosing=0
      counter = 0;
      stopwriting = False
      while stopwriting == False:
            if ser.in_waiting > 0:
                  #print("Serial Comms Waiting transmission")
                  line =
ser.read(2048)#ser.readline()#ser.readline().decode('utf-8').rstrip()
                  #print("got line")
                  print(line)
                  file1.write(line)
                  file2.write(line)
                  GPIO.output(LED, LEDstatus)
                  if LEDstatus ==1:
                        LEDstatus=0
                  else:
                        LEDstatus=1
                  counterforclosing=counterforclosing+1
                  if counterforclosing>10:
                        file1.close()
                        file2.close()
                        file1=open(logfile,"a")
                        file2=open(binfile,"a")
                        counterforclosing=0
```

APPENDIX C: CONSISTENT RECORD VALIDATION CODE

```
import cv2
import numpy as np
import os
import csv
import shutil

logDirectory = "J:/Video Files/15MAR22/VLog"
errorTracker=0


solutionfile='RTKRoverOrig'
corrSol='RTKRover'
correctedSolution=os.path.join(logDirectory,corrSol)
solname=os.path.join(logDirectory,solutionfile)
solf=open(solname, 'r', encoding='utf-8',errors='replace')
writer=open(correctedSolution,'w',newline='')
solfileread=solf.read().splitlines()
counterHour=0
counterMin=0
counterSec=0
timeHour=0
timeMin=0
timeSec=0

print(solfileread)
for idx,lines in enumerate(solfileread):
    print(idx, lines)
    individualLines=lines.split()
    if idx<25:
        writer.write(lines+'\n')

    if idx==25:
        timecapture = individualLines[1]
        timeForCounter = int(timecapture[0:2] + timecapture[3:5] +
timecapture[6:8])
        print(timeForCounter)
        counterHour = int(timecapture[0:2])

        counterMin = int(timecapture[3:5])

        counterSec = int(timecapture[6:8])

    if (idx>24):
        print(individualLines[1])
        timecapture=individualLines[1]
        timeInt=int(timecapture[0:2]+timecapture[3:5]+timecapture[6:8])
        print(timeInt)
        timeHour=int(timecapture[0:2])
        print(timeHour)
        print(counterHour)
        timeMin=int(timecapture[3:5])
        print(timeMin)
        print(counterMin)
        timeSec=int(timecapture[6:8])
        print(timeSec)
        print(counterSec)

        if(counterHour != timeHour or counterMin != timeMin or counterSec
!=timeSec):
            print('ERROR')
            spacerNeeded=1
            errorTracker=errorTracker+1
        if(counterHour == timeHour and counterMin == timeMin and counterSec
==timeSec):
            print('CORRECT')
            spacerNeeded=0
            writer.write(lines + '\n')


        if(spacerNeeded==0):
            if(counterSec==59):
                counterSec=0
                counterMin=counterMin+1
            else:
                counterSec=counterSec+1
            if(counterMin==60):
               counterMin=0
               counterHour=counterHour+1
            if(counterHour==24):
                counterHour=0
        if(spacerNeeded==1):
            #write the counter time to the file, increase counter, check the
log time again, rinse lather repeat.
            #Stopped here for the night
            print(counterHour)
            correctionRequired=1
            while(correctionRequired==1):
                writer.write(str(individualLines[0])+' '+
str(counterHour)+':'+str(counterMin)+':'+str(counterSec)+'.000'+'
'+ '\n')
                if (counterSec == 59):
                    counterSec = 0
                    counterMin = counterMin + 1
                else:
                    counterSec = counterSec + 1
                if (counterMin == 60):
                    counterMin = 0
                    counterHour = counterHour + 1
                if (counterHour == 24):
                    counterHour = 0
                if (counterHour != timeHour or counterMin != timeMin or
counterSec != timeSec):
                    print('ERROR after correction')
                    correctionRequired = 1
                if (counterHour == timeHour and counterMin == timeMin and
counterSec == timeSec):
                    print('CORRECT after correction')
                    correctionRequired = 0
                    writer.write(lines+ '\n')
                    if (counterSec == 59):
                        counterSec = 0
                        counterMin = counterMin + 1
                    else:
                        counterSec = counterSec + 1
                    if (counterMin == 60):
                        counterMin = 0
                        counterHour = counterHour + 1
                    if (counterHour == 24):
                        counterHour = 0
print("Total Number of Errors Found and accounted for:")
print(errorTracker)
```

The logic for record validation in Appendix C checks the RTK solution before use in model creation for consistent logs and replaces any missing datapoints with a correct time stamp and blank positional entry. After initializing the libraries and variables used, the log file storing the post-processed RTK solution is accessed. Each line of the file is scanned and checked for the satellite time for each positional packet. The informational header of the file is skipped over and is not needed for this step. The satellite times in each line are verified to be in numerical order, with no lost data segments present. If a missing positional packet is discovered, the code will create a timestamp for the missing data and inject a line item with blank positional data fields. This step is important for ensuring that each image chosen in the image syncing stage is properly assigned to the correct timestamp and positional data. Missing logs would cause the images to be assigned to incorrect timestamps and as a result the positional data tagged to each image would be flawed. After the missing positional data is accounted for with a blank entry the code moves on to the next line and checks it for continuity. The logic will operate in a loop until the entire RTK solution has been validated for integrity. An error tracker is output at the end of the file to display the total number of errors corrected.

APPENDIX D: IMAGE SORTING, SELECTION, LED LOGGING, POST PROCESSING

```
import cv2
import numpy as np
import os
import csv
import shutil
ppsvalidation=240
b=0
g=0
r=0
directory = "J:/Video Files/15MAR22/images"
logDirectory = "J:/Video Files/15MAR22/VLog"
exportdirectory = "J:/Video Files/15MAR22/AgisoftExport"
csvname='Log.csv'
correlationfilename='CoordinatedImagesTimesAndPositionsOfCamera.csv'
correlationpathname=os.path.join(logDirectory,correlationfilename)
FramesBetweenPPS=0
framestr=""
avgFrames=0
logname=os.path.join(logDirectory,csvname)
roverfile='Rover.txt'
rovername=os.path.join(logDirectory,roverfile)
trimmedRTKname='TimeClippedRTKSolution.txt'
trimmedRTKNamePath=os.path.join(logDirectory,trimmedRTKname)
trimmedRTK=open(trimmedRTKNamePath,'w',newline='')
trimmedRTKWriter=csv.writer(trimmedRTK)
correlationfile=open(correlationpathname,'w',newline='')
correlationfilewriter=csv.writer(correlationfile)
imagepositonnumber=0
RTKReadCounter=0
altLocation=9


import numpy as np
filenumber =0
#f=open(logname,'w',newline='')
solutionfile='RTKRover'
solname=os.path.join(logDirectory,solutionfile)
solf=open(solname, 'r', encoding='utf-8',errors='replace')
roverf=open(rovername, 'r',encoding='utf-8',errors='replace')
startRecordingTimestamp=roverf.readline()
print(startRecordingTimestamp)
tempTimeStorage=startRecordingTimestamp[15:21]
formatedStartTimeSearch=tempTimeStorage[:2]+':'+tempTimeStorage[2:4]+':'+temp
TimeStorage[4:]+'.000'
print(formatedStartTimeSearch)
solfileread=solf.read().splitlines()

print(solfileread)
for idx,lines in enumerate(solfileread):
    print(idx, lines)
    if formatedStartTimeSearch in lines:
        print("Start Time Found In RTK Solution")
        RTKStartLocation=idx
        break
```

```
for lines in solfileread[RTKStartLocation:]:
    trimmedRTKWriter.writerow([lines])
trimmedRTK.close()


RTKData=open(trimmedRTKNamePath,'r',encoding='utf-8',errors='replace')
RTKRead=RTKData.read().splitlines()


PPSFrames=0
fivemsLED=0
startRecordingLED=0
writer=csv.writer(open(logname,'w',newline=''))
bufferBetween=0
LEDLitCounter = int(0)
LEDTrackingFramesBetween=0
ppsCounting = 0
startRecCounting = 0
LEDValues=[]
binarystr=[]
timesBlinked=0
startingTimeSatImage="Void Right Now"
writer.writerow(["filename", "filenumber", "LEDValues", "pps","StartLED"])
#This will display all the available mouse click events

def flip_num(my_nu):
   return '1' if(my_nu == '0') else '0';

def gray_to_binary(gray):
   binary_code = ""
   binary_code += gray[0]
   for i in range(1, len(gray)):

      if (gray[i] == '0'):
         binary_code += binary_code[i - 1]
      else:
         binary_code += flip_num(binary_code[i - 1])

   return binary_code
events = [i for i in dir(cv2) if 'EVENT' in i]
print(events)

#This variable we use to store the pixel location
refPt = []
numTimesClicked = 0
clickedLocationArray = []
#click event function
def click_event(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:
        global numTimesClicked
        global clickedLocationArray
        print(x,",",y)
        refPt.append([x,y])
        font = cv2.FONT_HERSHEY_SIMPLEX
        strXY = str(x)+", "+str(y)
        cv2.putText(img, strXY, (x,y), font, 0.5, (255,255,0), 2)
        cv2.imshow("image", img)
        print("The clicked button is hit ", numTimesClicked)
        numTimesClicked=numTimesClicked+1
        clickedLocationArray.append([x,y])

    if event == cv2.EVENT_RBUTTONDOWN:
        blue = img[y, x, 0]
        green = img[y, x, 1]
        red = img[y, x, 2]
        font = cv2.FONT_HERSHEY_SIMPLEX
        strBGR = str(blue)+", "+str(green)+","+str(red)
        cv2.putText(img, strBGR, (x,y), font, 0.5, (0,255,255), 2)
        cv2.imshow("image", img)


#Here, you need to change the image name and it's path according to your
directory
img = cv2.imread("J:/Video Files/15MAR22/TestMask/TestMask.jpg")
img = cv2.resize(img, (1950, 1080))
cv2.imshow("image", img)

#calling the mouse click event
cv2.setMouseCallback("image", click_event)

cv2.waitKey(0)
cv2.destroyAllWindows()
print("Location Values Recorded are ")
print(clickedLocationArray)
currentSecondImage = "filename"
currentRecImage = "filename"
#start of second set of code

for filename in os.listdir(directory):
    tempname=os.path.join(directory,filename)
    #print(tempname)
    try:
        image = cv2.imread(tempname)
        print("New Image Successfully Read ")
    except:
        print("The image wasn't there")
    image = cv2.resize(image, (1950, 1080))
    try:
        LEDNumber=0
        LEDCount=0
        LEDValues=[""]
        for LED in clickedLocationArray:

            x=LED[0]
            y=LED[1]
            b=image[y,x,0]
            g = image[y, x, 1]
            r = image[y, x, 2]
            RGBVal = [r, g, b]
           # print("RGB Values for LED Number {}, picture number
{}".format(LEDNumber, filename ))
            #print([RGBVal])
            if LEDCount==0:
                LEDValues[0]=RGBVal
            if LEDCount>0:
                LEDValues.append(RGBVal)
            LEDNumber=(LEDNumber+1)
            LEDCount=LEDCount+1
    except:
        print("Whoops pixel snag failed.")


    LEDCount=len(LEDValues)
    print("There are this many LEDS: ", LEDCount, " for this file:
",filename)
    print(LEDValues)
    binarystr = [0,0]
    bintest=0
    LEDRGBCount=0
    ppsLEDValue=0
    startRecord=0


    for LEDRGB in LEDValues:
        print("This is the value being evaluated", LEDRGB)
        if LEDRGBCount==0:
            print("Checking Results of Start Recording LED")
            if LEDRGB[0] < 100 and LEDRGB[1] > 150 and LEDRGB[2] > 150:
                if startRecCounting == 0:
                    startRecCounting = startRecCounting+1
                    currentRecImage = filename
                    timesBlinked=timesBlinked+1
                    if timesBlinked ==2:
                        startingTimeSatImage=currentSecondImage
                        positionalRTKLine=RTKRead[0].split(" ")
                        print(positionalRTKLine)
                        if positionalRTKLine[10]=='':
                            altLocation=9
                        if positionalRTKLine[10]!='':
                            altLocation=10
                        if positionalRTKLine[9]=='' and
positionalRTKLine[10]=='':
                            altLocation=11


#correlationfilewriter.writerow([currentSecondImage,positionalRTKLine[4],posi
tionalRTKLine[6],positionalRTKLine[11],'#'+str(positionalRTKLine[:2])])
                        correlationfilewriter.writerow([currentSecondImage,
positionalRTKLine[4], positionalRTKLine[6], positionalRTKLine[altLocation],
'0.03', '0.03', '0.06'])
                        RTKReadCounter=RTKReadCounter+1
                        copyfrom = os.path.join(directory,
currentSecondImage)
                        copyto= os.path.join(exportdirectory,
currentSecondImage)
                        shutil.copy(copyfrom,copyto)
                else:
                    startRecCounting = startRecCounting+1
                startRecord = 1
                print("it was high")
            else:
                startRecord=0
                print("It was low")
                startRecCounting = 0
        elif LEDRGBCount == (len(LEDValues)-1):
            print("Checking PPS Value ", LEDRGBCount)
            if LEDRGB[0] > 200 and LEDRGB[1] > 200 and LEDRGB[2] <250:
                if ppsCounting == 0:
                    ppsCounting = ppsCounting + 1
                    currentSecondImage = filename

                    if timesBlinked > 1 and ppsvalidation>200:
                        ppsvalidation=0
                        positionalRTKLine = RTKRead[RTKReadCounter].split("
")
                        RTKReadCounter=RTKReadCounter+1
                        print(positionalRTKLine)
                        if positionalRTKLine[10]=='':
                            altLocation=9
                        if positionalRTKLine[10]!='':
                            altLocation=10
                        if positionalRTKLine[9]=='' and
positionalRTKLine[10]=='':
                            altLocation=11
                        correlationfilewriter.writerow([currentSecondImage,
positionalRTKLine[4], positionalRTKLine[6], positionalRTKLine[altLocation],
'0.03', '0.03', '0.06'])
                        copyfrom = os.path.join(directory,
currentSecondImage)
                        copyto= os.path.join(exportdirectory,
currentSecondImage)
                        shutil.copy(copyfrom,copyto)
                        correlationfile.close()
                        correlationfile = open(correlationpathname, 'a',
newline='')
                        correlationfilewriter = csv.writer(correlationfile)
                else:
                    ppsCounting = startRecCounting + 1

                ppsLEDValue=1
                print("it was high")

            else:
                LEDLogic = 0
                ppsLEDValue=0
                print("It was low")
                ppsCounting = 0
                ppsvalidation = ppsvalidation + 1
        else:
            print(LEDRGBCount)
            print("Nothing fit")
        LEDRGBCount = LEDRGBCount + 1


    pps=ppsLEDValue
    startRecordingLED=startRecord
    print(pps)
    print(startRecordingLED)
    writer.writerow([filename, filenumber, LEDValues, pps,
startRecordingLED,currentSecondImage, currentRecImage,startingTimeSatImage,
ppsvalidation])
    filenumber=filenumber+1

correlationfile.close()
copyfrom = os.path.join(logDirectory, correlationfilename)
copyto = os.path.join(exportdirectory, correlationfilename)
shutil.copy(copyfrom, copyto)
```

The logic in Appendix D handles a critical stage of the process. The coordination of images to proper positional data as close to the instant that the positional message was received is accomplished with this code. Video recorded during data capture is segmented out into individual images representing each frame of the video beforehand. The images are scanned sequentially the states of the indicator LEDs showing PPS and recording status are logged. The image closest to the instant new positional data was received is selected and sorted into a separate folder. A log of the position at the instant the data was received is coordinated to these images. This final log is used for geo-referencing the selected images in the Agisoft Metashape program.

After initialization, the image directory, log storage locations and the RTK solution file are accessed. The file created during data recording on the rover indicating the starting time of the RTK data that relates to the images is accessed and the starting time is stored for later use. The RTK solution is scanned until the starting time is found. This will be the time and positional data matched to the image showing when the positional data was received. A new logfile is created to store the coordinated filenames, timestamps, latitude, longitude, and altitude for geo-referencing. Next, the test image will be examined. A test image is selected manually from the images beforehand which shows the start recording and PPS LEDs lit simultaneously. The test image is displayed on the screen, and location recordings are made of mouse clicks entered by the user. The first mouse click designates the center of the start recording LED. The second mouse click represents the center of the PPS LED. These locations are saved for scanning images for LED status. For validation by the user, the number of LEDs as input by mouse clicks is displayed. Each image is then examined at the defined LED locations and using a threshold value, is defined in terms of illumination. Each image is logged with the LED values with the first detected PPS image of each second moved to a separate folder. Each moved image is also tied to a positional data entry and logged. Finally, the completed geo-referencing file is copied to the image directory as well. These images and log file will be used by Agisoft Metashape for geo-referencing.