ALIS

Aerial Land Inspection System

Final Report

Vermeer Corporation

Team Members
Brian Gillenwater
Nathan Kent
Quinn Murphy
Bryce Poellet
Jonathan Schlueter

April 24, 2016

Table of Contents

List of Figures

Introduction

Purpose

The Aerial Land Inspection System (ALIS) is a solution to remotely and autonomously map the terrain of a future work site from the air. Once completed, a three-dimensional terrain map will be generated which will allow people to view the terrain interactively through a head-mounted display.

Scope

When given a location and area, ALIS is responsible for generating and executing a quadcopter flight path. From this flight path, ALIS captures images to generate a 3D model of the terrain. This model is uploaded into a game engine that allows exploration with a head-mounted display. The user is responsible for knowing the area and size of the desired location. The documentation provided here details the design decisions made while developing the entire ALIS system. These details include: the system description with use cases, constraints, and assumptions; functional requirements and non-functional requirements; and the top-level system design.

Definitions

Quadcopter - A rotorcraft with four propellers.
Photogrammetry - The use of multiple two-dimensional photographs of a subject to reconstruct the third-dimension.
Virtual Reality System - A piece of multimedia used to simulate physical presence in a virtual environment.
Oculus Rift - A head-mounted display that renders stereoscopic images to produce a 3D environment.

Acronyms

ALIS - Aerial Land Inspection System
FAA - Federal Aviation Administration

Problem and Needs

As it currently stands, examining a job site is an expensive and time-consuming task. It requires sending someone out to the area and having them take notes and photographs. This process can potentially involve several days' worth of time and work. By finding a way to automate this process, the cost of inspecting an area will dramatically decrease and the ability to view the designated area will not be limited to the individuals physically sent to the work site.

Overview

ALIS is intended to enable the mapping of a worksite remotely. To accomplish this, a quadcopter equipped with a high resolution camera is sent out to autonomously take pictures of the worksite from the air. By taking enough images, photogrammetry techniques can be applied to the images to generate a three-dimensional model. This model is then sent to a virtual reality system allowing people to examine the worksite without needing to travel there. ALIS requires very little user interaction and operates under its own control as much as possible.

Deliverables

The system includes a quadcopter equipped with a high resolution camera (a DJI Phantom 3 Advanced), a powerful Windows PC, and an Android device used for quadcopter control. The quadcopter captures aerial photos and transfers them to the PC application. The system will then use photogrammetry techniques to generate a high-resolution terrain model. This model is then sent to a virtual reality system allowing people to examine the worksite without needing to travel there. In summary:

- A PC application capable of generating/communicating a flightpath and receiving images
- A high-resolution model generated from aerial photography
- A game engine project including this model, viewable in a head-mounted display

Design

Functional Requirements

- Sustained flight in adverse weather
- At least 20 minutes of flight time
- Fly up to ½ mile away from controller
- Take images with more than 50% overlap
- Model generated in less than 6 hours
- Model is viewable in a virtual reality platform

Non-Functional Requirements

- User interaction with the system shall be simple and error free
- A clear and accurate model will be generated, regardless of terrain
- The device shall work in clear and windy environments (within reason)
- The system shall not crash or become unstable during flight
- The model will be viewable in a virtual reality platform
- The system will be safe to operate, and include emergency stop controls

Functional Decomposition

The system is started by the user launching the PC application and selecting a location to be mapped. Parameters can then be set that control the area's size and shape, as well as quadcopter altitude. The user then builds a map, which will generate a set of waypoints that will appear on the built-in map interface. Once the user is happy with the generated flight path, the user will select 'Upload'. After selecting a device and confirming, the flight plan will be uploaded to the Android platform wirelessly.

After receiving the waypoint data, the Android application transforms the flight path into a format the quadcopter can understand, and passes this information to the quadcopter. The quadcopter executes the requested flight, taking pictures along its route. Once it returns, images are transmitted to the PC application.

After receiving all images, the PC application passes these images to the photogrammetry software. This software outputs an .obj model file. Once the model is generated, the model is imported into a game engine, and a project is generated that supports the use of a head-mounted display. See Figure 1 below for a pictorial description of the overall system. Figure 2 offers a more in-depth look at the system.

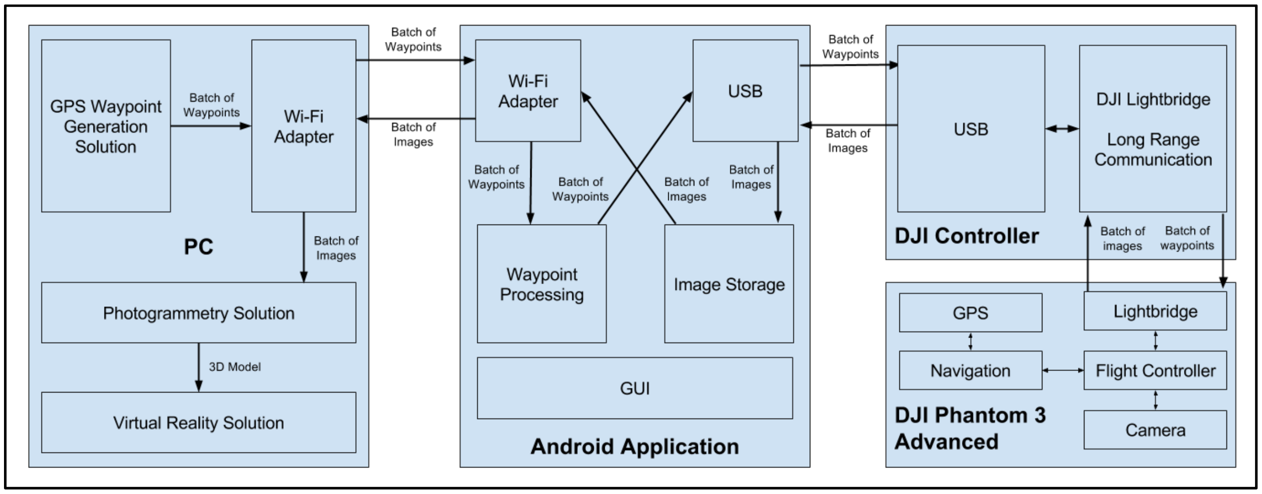


Figure 1: Functional System Diagram

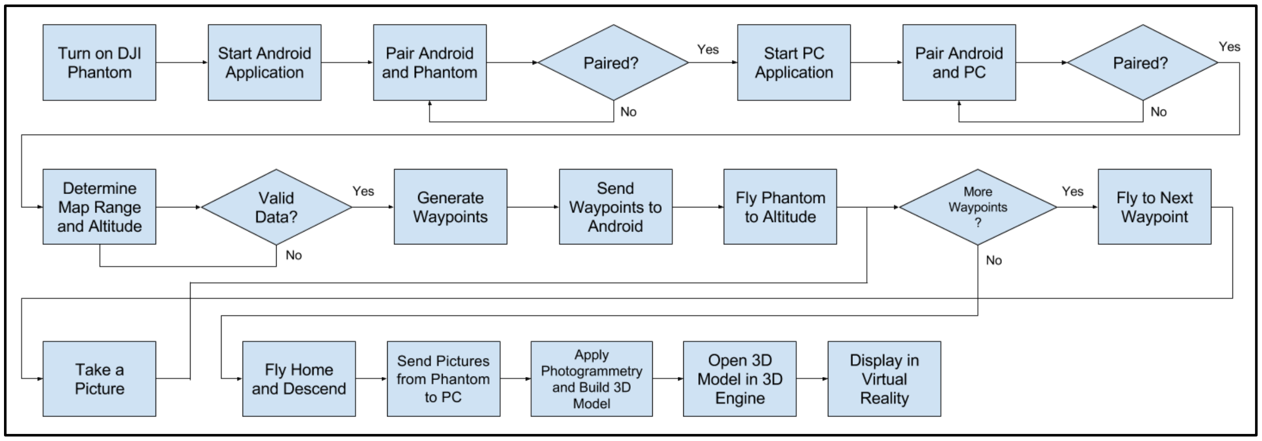


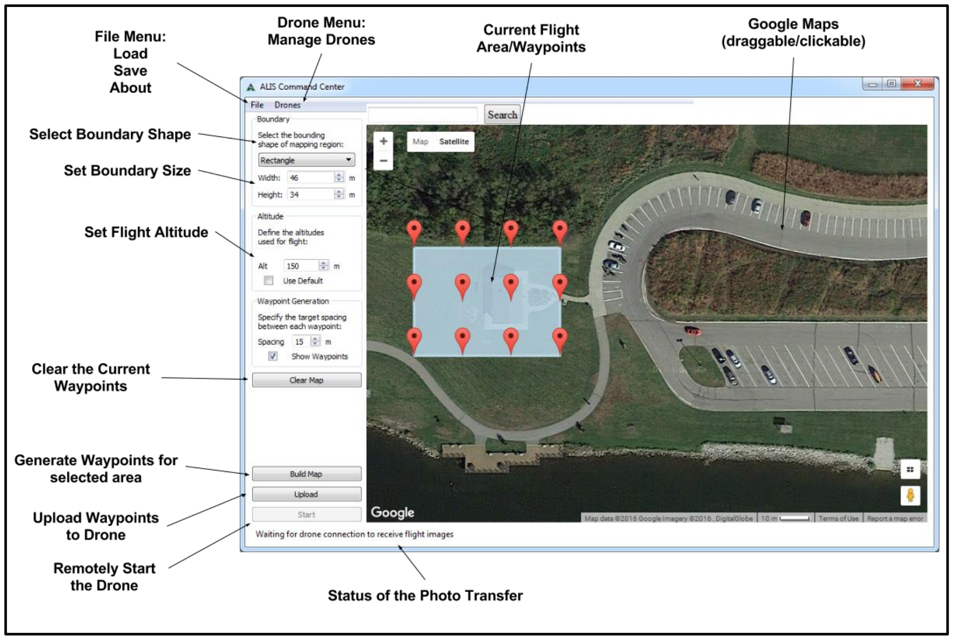
Figure 2: State Diagram

Screen Sketches

Two applications were developed for users of the ALIS system. The first is a PC application to select mapping areas and the second an Android application to fly the quadcopter. Our team developed screen sketches and then UI mockups to present to the client and to help with the development process.

ALIS Command Center

ALIS Command Center, our Windows application, handles user configuration of the flight. Figure 3 shows the main page users are presented with when they launch the Command Center. Parameters are listed in the sidebar, and drop-down menus provides system control commands such as loading and saving a generated map.


File Menu: Load Save About
Drone Menu: Manage Drones
Current Flight Area/Waypoints
Google Maps (draggable/clickable)
Select Boundary Shape
Set Boundary Size
Set Flight Altitude
Clear the Current Waypoints
Generate Waypoints for selected area
Upload Waypoints to Drone
Remotely Start the Drone
Status of the Photo Transfer
ALIS Command Center
File
Drones
Boundary
Altitude
Waypoint Generation
Clear Map
Build Map
Upload
Start
Search
Map
Satellite
Google

Select the bounding shape of mapping region: Rectangle
Width: 46 m
Height: 34 m
Define the altitudes used for flight:
Alt 150 m
Use Default
Specify the target spacing between each waypoint:
Spacing 15 m
Show Waypoints ✓
Waiting for drone connection to receive flight images

Figure 3: Command Center Main Window

Figure 4 shows the drone management screens of the Command Center. After selecting ‘Manage Drones’ from the Drones drop-down menu, the user can choose to add a new drone, delete an existing drone, or edit the current configuration of a drone. Each drone has a name, IP address, and password associated with it.

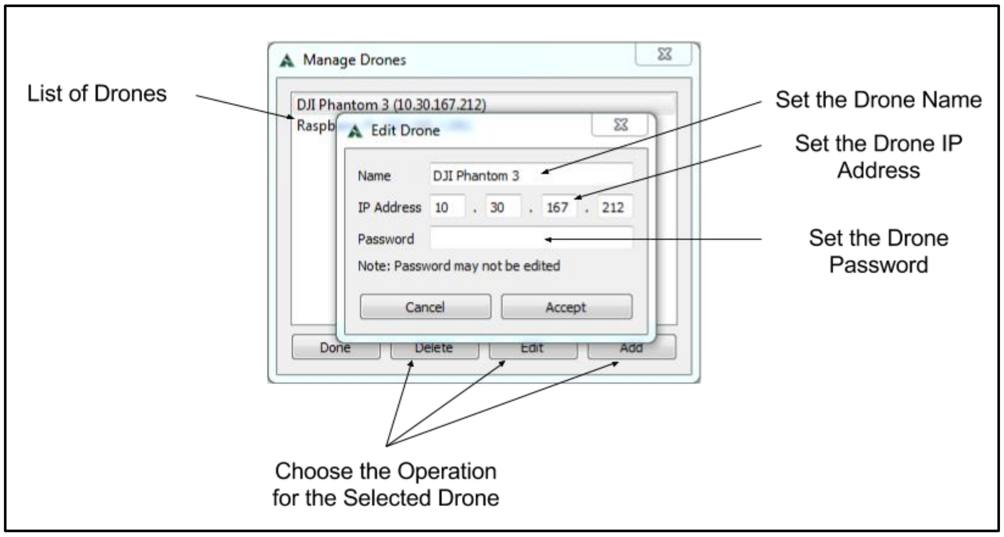


Figure 4: Command Center Edit Drone Window

Figure 5 shows the upload window users are presented with when they click ‘Upload’ on the main window. This allows users to select an Android device, and upload the flight plan over a network connection.

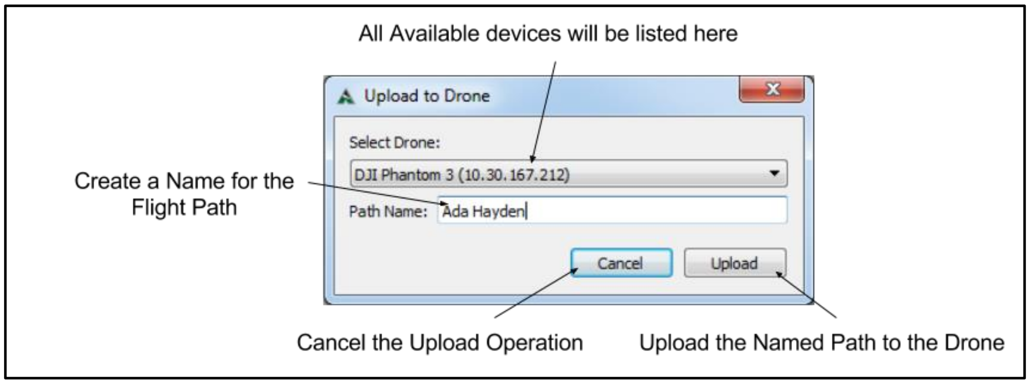

All Available devices will be listed here
Upload to Drone
Select Drone:
DJI Phantom 3 (10.30.167.212)
Create a Name for the Flight Path
Path Name:
Ada Hayden
Cancel
Upload
Cancel the Upload Operation
Upload the Named Path to the Drone

Figure 5: Command Center Upload Window

ALIS LITE

ALIS LITE (Location and Image Transfer Entity), the Android application, is responsible for controlling the quadcopters flight plan and transferring images back to the PC. Figure 6 shows the first screen that users who use ALIS LITE will see. On the top right are different functions that the user can use before actually starting the flight. From left to right, they allow the user to manually upload a binary file of the waypoints, view those waypoints with Google Maps, change the settings of the application, and calibrate the compass. Pressing start will send the quadcopter off to do its mission.

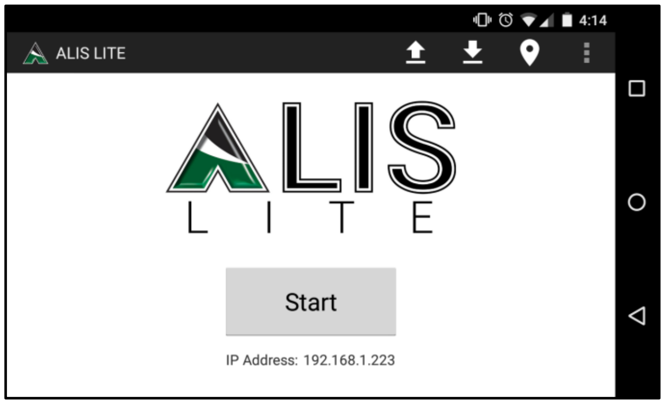


Figure 6: ALIS LITE Connection Screen

Figure 7 shows the waypoint verification screen. The app will show the loaded waypoints on a map. This allows the operator to verify the correct mission is loaded and allows verification of the quadcopters coverage area.

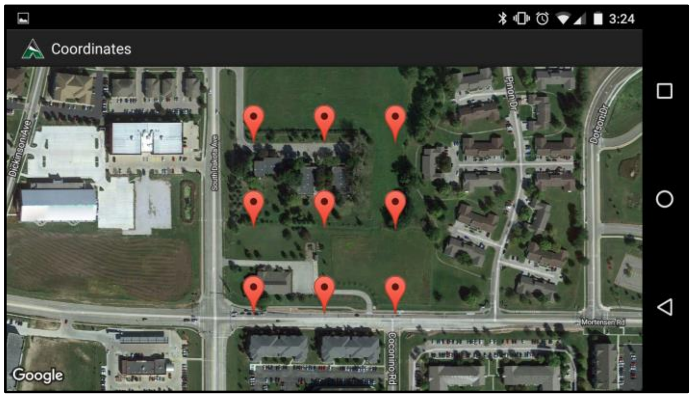
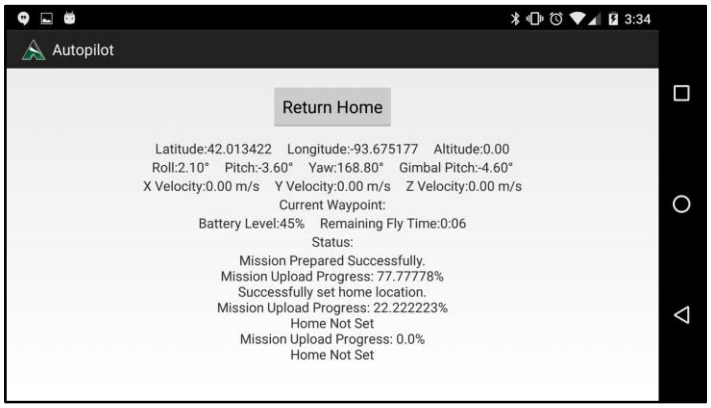


Figure 7: ALIS LITE Waypoint Screen

Figure 8 shows the screen that is shown during a mission. It reports back the status of the quadcopter (location, attitude, battery life, mission status) as well as printing out any status messages like errors or updates in a scrolling list.



Latitude:42.013422 Longitude:-93.675177 Altitude:0.00
Roll:2.10° Pitch:-3.60° Yaw:168.80° Gimbal Pitch:-4.60°
X Velocity:0.00 m/s Y Velocity:0.00 m/s Z Velocity:0.00 m/s
Current Waypoint:
Battery Level:45% Remaining Fly Time:0:06
Status:
Mission Prepared Successfully.
Mission Upload Progress: 77.77778%
Successfully set home location.
Mission Upload Progress: 22.222223%
Home Not Set
Mission Upload Progress: 0.0%
Home Not Set

Figure 8: ALIS LITE Status Screen

Implementation Details

Networking Specification

The Windows application and Android application communicate over a Wi-Fi network (IEEE 802.11 b/g/n) using TCP/IP. The Android application runs as a server for waypoint transfer, and the Windows application runs as a server for photo transfer. The current protocol validates both the client and server using a salted passphrase stored on each device. Figure 9 shows a typical network transaction. This transfer may be encrypted, but for the prototype implementation it is not.

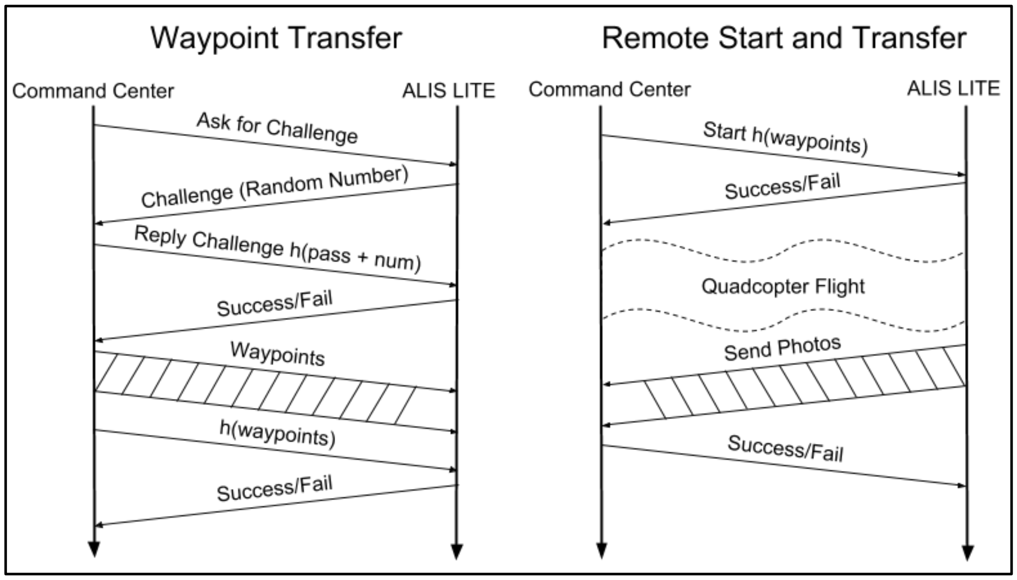


Figure 9: Network Message Order

Message Descriptions

Every message passed between the Windows application and the quadcopter will start with a single byte that indicates the contents of the rest of the message. Following the message identifier is the rest of the message, the contents of which are specific to each message. Table 1 specifies each type of message and the associated identifying prefix, message contents, and message direction. All message contents that require more than a single byte must be sent in big endian order.

Table 1: Networking Message Types

Identifying Byte	Message Description	Message Size (Bytes)	Direction
0xF*	Success	1	Both
0x7*	Failure	1	Both
0x00	Request Verification Challenge	1	PC -> Android
0x01	Verification Challenge	5	Android -> PC
0x02	Verification Response	33	PC -> Android
0x03	Waypoint Transfer	3 + 23 * num_waypts	PC -> Android
0x05	Start Flight	33	PC -> Android
0x08	Photo Transfer	35 + variable	Android -> PC

Request Verification Challenge (0x00)

No Additional Data
The client requests a random number to include in the verification key. This message must be sent before any other messages are accepted.

Verification Challenge (0x01)

int32(Random Number)
The server replies to the verification challenge. A random integer is returned to be included in the verification key.

Verification Response (0x02)

32-bytes(SHA-256 Hash)
The hashed key is combined as shown with the random number from the server.

$$h_{SHA-256}\left(randomNumber + h_{SHA-256}(verificationKey)\right)$$

This is then hashed together and sent back to the server. This is done to prevent replay attacks involving the hashed verification key. Once the client replies with the correct key the system is trusted and waypoint transfer can begin.

Waypoint Transfer (0x03)

int16 (Number of Waypoints) + 23*N-bytes (Waypoint Data) + 32-bytes (SHA-256 Hash)
The waypoint transfer will send the number of waypoints to be transferred, followed by the waypoints. The waypoints are transmitted according to Table 2. Finally, a 32-byte hash of the waypoint data is sent to verify the integrity of the waypoints.

Table 2: Waypoint Byte Order

Description	Byte Offset	Data Type	Minimum	Maximum
Latitude of the Waypoint	0	f64	-90	90
Longitude of the Waypoint	8	f64	-180	180
Altitude of the Waypoint (meters)	16	f32	0	120
Heading of the Quadcopter (degrees)	20	i16	0	360
Pitch of the Gimbal (degrees)	22	i8	-90	30

Start Flight (0x05)

32-bytes (SHA-256 Hash)
The PC application can request the quadcopter to be started remotely. This is done by including the hash of the waypoints last uploaded to the quadcopter to ensure that the expected flight is being started.

Photo Transfer (0x08)

32-bytes (SHA-256 Hash) + 2-bytes (Number of Photos) + X (Size of Photo Stream)
The quadcopter is sending flight photos back to the PC application. After the identifying byte, the quadcopter includes the 32-byte, SHA-256 hash of the waypoints corresponding to the flight it has completed. The quadcopter then sends a 2-byte value indicating the number of photos to be transferred. The photo transfer then follows with each photo byte stream being preceded by a four-byte value indicating the size of the following photo.

Success (0xF*)

No Additional Data
This is used to send success messages from either system. Any message starting with 0xF can include a 0-15 value indicating the type of success.

Error (0x7*)

No Additional Data
This is used to send error messages from either system. Any message starting with 0x7 can include a 0-15 value indicating the type of error.

UI Specifications

ALIS Command Center

The Windows application is where the majority of interaction and processing happens for ALIS. A user is able to do the following:

- Waypoint Planning & Generation
    - Define a flight altitude (can use defaults if unknown)
    - Build a waypoint map
    - View the generated map and waypoints with a built-in map utility
    - Save the current map and its waypoints
    - Load a previously saved map
- Communication & Control
    - Select a recognized Android device and upload the current map
    - Verify a successful map transfer to the device
    - Remotely start the quadcopter mission

ALIS LITE

Because the Android application is a thin-client, the user interface is minimal and mostly comprised of debug information. The user is able to do the following things:

- Perform an emergency landing of the quadcopter
- Force the quadcopter to return to home base
- Start the quadcopter flight sequence
- View the status of the transfer of data points from Windows to Android
- View the status of the transfer of images from the quadcopter to Android
- View the status of the transfer of images from Android to Windows
- View errors associated with any data transfer
- View the waypoints that have been loaded onto the quadcopter
- View the current location, altitude, speed, and attitude of the quadcopter
- View the current battery life of the quadcopter

Hardware/Software Specifications

- Software Libraries and Version Info
  - ALIS Command Center (Windows Application)
    - C++11
    - Qt5 Cross-Platform UI Framework v5.5
    - Runs on Microsoft Windows 7 and later
  - ALIS LITE (Android Application)
    - DJI Mobile SDK 3.0.1
    - Runs on Android 4.2 and later
- The PC and Android device need to be on the same network.
  - Communication happens over port 3850.
  - The software prompts to open a firewall port.

Testing

Testing Process

Developing a testing plan for ALIS was complicated due to the multiple software systems involved. We decided to mainly focus on integration testing between the systems. Each system was tested using mock inputs.

ALIS Command Center

Due to the Command Center primarily being a user interface, there was no good way to programmatically test features. Instead, we implemented compartmentalized testing. First, we ensured waypoint generation worked flawlessly. Next, we added UI fields to the process allowing them to make changes to the generated waypoint pattern, eventually we enabled full user control. To verify correctness, we manually inspected the output data from many map sizes and waypoint heights.

Testing the Windows Application involved the following:

- Checking generated flight path for accuracy and user parameter inclusion
- Make sure any generated waypoints fully comply with FAA regulations
- Checking that flight path covers the specified area sufficiently for the photogrammetric reconstruction to be successful
- Checking that the flight path is transmitted to the Android Application in the correct format and with the correct data

ALIS LITE

The Android application is a thin client, so it doesn't have a lot of its own algorithms or methods that need to be tested. Furthermore, because most of the actions it performs require communication with the quadcopter, automated testing is close to impossible. The networking code was tested for all kinds of improper input such as incorrect messages coming in at random times or disconnecting in the middle of a transfer. The flight system itself was tested on multiple occasions as we went out to capture more photos from the sky. Additionally, Twitter's Fabric framework was used in ALIS LITE (specifically Crashlytics) which reports the stack trace of any crash of the application to the cloud in order to find the source of bugs when not connected to a computer.

Networking

The networking protocol was tested in two steps. First, for each system (Windows and Android) we implemented the desired networking functionality, and then began testing only that system by sending and receiving packets to a custom application written specifically for these tests. This method allowed both systems to undergo the initial development and testing simultaneously without waiting on the other system to finish development or fix bugs. The custom testing application also allowed us to test the networking code for error recovery that would have been hard to replicate using only the end systems.

Once we had tested the two systems in isolation, we brought the two together and began testing the systems with each other. This final testing involved attempting to send actual waypoint sets or photo sets between the systems and making sure that they were received correctly. This testing also allowed us to find any ambiguities in our protocol which caused our tests to fail because an assumption was made while developing one system, but was not made during the development of the other system.

Photogrammetry

To test the captured photos, we first verified that a model could be generated from the photo set. Once the model was generated, we visually inspected the output in order to check for accuracy and clarity. Due to the nature of the problem combined with the limitations in our chosen photogrammetry solution, we were not able to automate the testing of model generation.

Testing the Photogrammetry process involves the following steps:

- Loading the pictures into the photogrammetry software
- Perform feature detection and feature pair matching on the photos
- Converting the pictures into a point-cloud
- Transforming the point-cloud into a textured model usable in Unity
- Import into Unity to confirm that it accurately depicts the area in question

Testing Results

To evaluate the results, we needed to separate the evaluation of the photo capture portion from the evaluation of the photogrammetry portion. The ALIS system was able to generate and transmit waypoints over the network and the quadcopter was able to fly to the specified GPS waypoints. During the flight, photos were taken and stored on the quadcopter's SD card. Upon landing, the images were transferred to the PC. This chain of processes worked successfully and so the photo capture system passed our tests.

To evaluate the photogrammetry, we needed to observe the model using an Oculus Rift. We noted that the software had difficulty handling groups of trees. However, the software was able to produce models that gave a reasonably accurate representation of the designated area.

Appendix I - Operation Manual

1. **Clone the GitHub Repo**
   At the moment, the repository for our code is https://github.com/QMurphy/ALIS.git. You will need to clone that repository in order to retrieve the ALIS Command Center and ALIS LITE code.
2. **Installing the ALIS Command Center**
   A. **Install Visual Studio 2013**
   The first step in installing the ALIS Command center is installing Visual Studio 2013. Other versions of Visual Studio may work, but we have only tested compilation on MSVC 12.0 so we recommend Visual Studio 2013.
   B. **Install Qt 5.5.1**
   You can find Qt 5.5.1 x86 for Visual Studio 2013 below:
   http://download.qt.io/archive/qt/5.5/5.5.1/qt-opensource-windows-x86-msvc2013-5.5.1.exe
   C. **Install the Visual Studio Add-in for Qt5**
   This will allow you to build Qt applications inside of Visual Studio:
   http://download.qt.io/official_releases/vsaddin/qt-vs-addin-1.2.5.exe
   D. **Set up system environment variables**
   You will then need to set the Qt path as a system environment variable by creating a variable called QTDIR and setting its value to the msvc2013 folder in your Qt folder (e.g. "C:\Qt\Qt5.5.0\5.5\msvc2013"). Modify the PATH system environment variable by adding "%QTDIR%\bin" to the list of directories.
   E. **Install OpenSSL**
   You will need version 1.0.2g of Win32 OpenSSL which is available below:
   https://slproweb.com/download/Win32OpenSSL-1_0_2g.exe
   F. **Build the ALIS Command Center**
   Once you open the Visual Studio solution, you should be able to build and run the ALIS Command Center. If you get an error stating that "there is no Qt version assigned to this project for platform Win32" then you will need to go to the Qt5 menu in Visual Studio and select Qt Options and add a new Qt version with the name "Qt5 - msvc2013" and a path

to wherever you set the QTDIR environment variable to. You will also need to make sure that this is marked as the default version.

3. **Installing ALIS LITE**
   - **A. Download Android Studio**
     Android Studio is the IDE and build system for ALIS LITE. You can download and install both it and the Android SDK from this link:
     https://developer.android.com/sdk/index.html
   - **B. Install to Android Device from Android Studio**
     You should be able to open the ALIS LITE directory as an Android Studio project and after the Gradle configuration finishes, you should be able to build and install the application to a connected Android device.
4. **Installing RealityCapture**
   - **A.** Obtain a License from https://www.capturingreality.com/
   - **B.** Download the software installer from your account
   - **C.** Install the software using the wizard.
5. **Installing Unity**
   - **A.** Go to https://unity3d.com/get-unity/download?ref=personal and download the latest installer for the Windows platform.
   - **B.** When asked to choose components, accept the defaults. Note that 'Windows Build Support' should be checked. You can add other build modules if you wish.
   - **C.** Advance through the windows and install to your desired location. Upon launching, you may be asked to create a free account for Unity
6. **Installing Oculus Rift Utilities**
   - **A.** First, make sure your computer meets the minimum requirements to run the Oculus Rift, listed at https://www.oculus.com/en-us/blog/powering-the-rift/. These are not hard requirements, but such hardware enables a more comfortable experience.
   - **B.** Go to https://www.oculus.com/en-us/setup/ and download the Oculus Runtime utility to your computer. Advance through the installer, accepting defaults.
   - **C.** Download and install the OVRPlugin, which allows the Unity game engine to work with the Rift. Go to https://developer.oculus.com/downloads/, find "OVRPlugin for Unity 5", and click "Details". Follow the directions that are listed on that page. Depending on your needs, you may need to install a different version of the Unity game engine.
7. **Using ALIS**
   - **A.** Before operating ALIS, be sure to read through the FAA guidelines for commercial drone flight (https://www.faa.gov/uas)
   - **B.** Place the DJI Phantom 3 Advanced on the ground and turn it on.
   - **C.** Connect the Android device with ALIS LITE installed on it to the DJI remote controller and turn the controller on. The Android device may ask if you would like to open an application, at this point you should select ALIS LITE.
   - **D.** Once ALIS LITE is running, you have the option to either load a preloaded map already on the device (by pressing the "up" arrow on the main screen" or accept one from the ALIS Command Center over the network.

E. From the ALIS Command Center, you can find the area that you would like to scan on the map interface and select corners of the area by right clicking. You can also set the altitude and spacing of the data points as well as fine-tuning the width and height.

F. Before you send the data points to ALIS LITE, you will need to configure the IP address of that device by going to Drones, Manage Drones, and the selecting Add and typing the relevant information. When that is done you will be able to build the map and then upload it to the device.

G. At this point, you should be able to start the quadcopter remotely as well as starting it manually from ALIS LITE.

H. Once the system is running, you simply need to wait for the quadcopter to visit all of the points. There is currently a bug in the DJI Mobile SDK that prohibits us from changing the base speed, though you can make the quadcopter go faster by holding forward on the right stick.

I. Once the quadcopter finishes its run and lands, it will begin transferring photos back to the Android device which will in turn transfer them back to the PC. This transfer is quite slow, however, and users may wish to simply connect the quadcopter directly to the PC to expedite transfer.

J. Open up RealityCapture and begin to create the model.

i. Select the **Workflow** tab.
ii. Add a folder to the project and select the images you captured.
iii. Click the align images button.
iv. Calculate a *Preview Quality* model, use the dropdown to select this.
v. Under the **Reconstruction** Tab select the *Set Reconstruction Region* button. Use the 3-D tool to select the model that you want generated in high quality. The markers above the model represent the locations of images that are captured.
vi. Under the **Workflow** tab using the Calculate Model dropdown select *High Quality* to now generate a high quality model of the selected region.
vii. It is recommended to save the project in this state and work with a copy of the project. There is a way to select which model is active but this seems to be faster and simpler.
viii. Next we will decimate the model. Under the **Reconstruction** tab select *Simplify Tool*. This tool will appear in the bottom left corner. Our team recommends between 500,000 and 1,000,000 polygons. Once you have entered the value choose simplify.
ix. Now select the *Smoothing Tool* and click smooth. This will reduce rough edges in the model.
x. Select the *Texture* button from the **Workflow** tab. This will generate the overlay image that will be used in the game engine.
xi. Under the “3. Export” area in the **Workflow** tab select Mesh.
xii. We recommend setting the compression to PNG if you intend to use Unity. This is done after choosing a save location and name.
xiii. Click OK. You should now have a model that is ready for import into the Unity game engine.

K. Import into Unity

i. Open the Unity game engine

ii. Create a new project. Give your project a name, and make sure the ‘3D’ option is highlighted
iii. Go to “Assets” > ”Import New Asset…” and select the PNG and OBJ you created from CapturingReality. This may take some time, as they are large files.
iv. In the Unity Editor, under the Project tab, select the PNG you just imported. In the Inspector tab, set the “Max Size” to be 8192. Also, make sure the ‘Texture Type’ is set to ‘Texture’. Hit ‘Apply’ at the bottom of the Inspector.
v. When you imported the OBJ, Unity generated a ‘Materials’ folder. Expand it and select the material inside. In the Inspector, click the small circle next to the ‘Albedo’ field, and select the PNG. This applies the texture to your model.
vi. Lastly, simply drag your mesh object from the Project explorer into the scene. If you want the model to have colliders (for a character to walk on, etc), click on the OBJ in the Project explorer, and in the Inspector tab, go to the ‘Model’ sub-tab, and check the ‘Generate Colliders’ box, and hit ‘Apply’.
vii. If you want Oculus support for this game, go to https://developer.oculus.com/downloads/ and select the ‘Oculus Utilies for Unity 5’. Once you download and import the asset into a project, it will be Oculus-ready.

L. If you want a pre-generated Unity project with Oculus support and gamepad movement control, please contact one of the authors.

Appendix II - Alternative Designs

Photogrammetry Software

Initially, we were planning on using an open-source application called CMP-MVS. However, we found that the models it produced were insufficient for the purpose of land inspection. Additionally, the process of converting the photographs into 3D models took much longer than we were hoping (over six hours) and the documentation was basically non-existent.

We then examined several other photogrammetry options, focusing largely on two products: Pix4D and RealityCapture. Pix4D is an established product with a well known record, but the RealityCapture had a similar feature set for a fraction of the cost. Based on our testing, both of the products produced high quality models quickly.

Network Security

For a period of time, we played with the idea of using public/private key pairs to do authentication between the server and the quadcopter. By digitally signing all communication between the two, we would ideally have tamper-resistant communication. However, we found that we were not able to develop a working solution in the limited time that we had remaining. This lead to us looking into implementing SSL/TLS for the communication, but it was decided that our time was best spent finishing up other aspects of the project.

Quadcopters

Several other quadcopters were considered for the project. The final list included the following:

- Lumenier QVA250 Kit with OpenPilot
- Parrot Bebop
- DJI Matrice 100
- DJI Phantom 3 Advanced

The team had to list the pros and cons of each product. Most of the options, besides the Phantom 3 advanced had major showstoppers. The DJI Matrice was too expensive for the project's budget with a list price of over $3000. The parrot Bebop had a front facing camera which was not compatible with the photogrammetry solution. The Lumenier quadcopter looked promising, however the OpenPilot project is currently unstable and does not have any documentation available.

Appendix III - Other Considerations

DJI SDK Constraints

While the Phantom 3 Advanced worked well for our project, there were some constraints in the SDK that required some workarounds from the ideal case. The primary constraint we ran into was that the SDK only allowed five photos to be taken at each geographical waypoint while the Windows Command Center was designed to capture nine photos at each waypoint. In order to meet the required nine waypoints, the Android thin client needed to modify the received waypoints by splitting each waypoint requiring more than 5 photos into two separate waypoints, with the second waypoint being shifted North by one meter.

FAA Drone Regulations

The FAA regulations has strict rules regarding the operation of drones, and to complicate the matter even further, these regulations changed in the middle of our project. We made sure that we stayed informed about the regulations as they changed so that we would know if our project was ever going to violate them. In the end, the only change that truly affected us was the need to register our drone halfway through the project.

Photography Pattern

Going into this project, we had no idea what was the best pattern to take the photos in to generate the best three-dimensional model. In the end, we settled upon a pattern that logically seemed ideal and in testing led to good models. This pattern involved taking nine photos at each geographical location; one at 45° down for each cardinal and ordinal direction and a final one straight down. We did try to reduce the number of photos by removing the ones in the ordinal directions and found that we could not generate a model, but we have not proof that our pattern is the most efficient for generating a model with the fewest number of photos.

Waypoint Spacing

Currently we allow the user the specify both the altitude the quadcopter will fly at, as well as the distance between all the waypoints. However, this has the potential to lead to waypoint sets that do not have enough overlap in the photos for the photogrammetry to successfully build the three-dimensional model. For example, if the user specifies a very large spacing, but a low altitude, there could be gaps in the photos from adjacent waypoints. We discussed this issue, and decided that we could determine the needed relationship between altitude and spacing, but the solution would be specific to the quadcopter and camera combination we were using. One idea we talked about was creating the ability to load different quadcopter profiles that would contain all the data needed to perform the altitude-spacing calculations, however we decided we did not have time to add this additional feature. We therefore decided to leave both the waypoint spacing and altitude under user control rather than tie the Windows application to being specific to the camera on the DJI Phantom 3 Advanced.

Appendix IV - Code and External Documentation

Source Code

All source code is located at the ALIS GitHub Repository: https://github.com/QMurphy/ALIS

Documentation

Qt Framework: http://doc.qt.io/qt-5/index.html
Android SDK: http://developer.android.com/reference/packages.html
DJI Android Mobile SDK: https://developer.dji.com/mobile-sdk/documentation/android