gRAID: A Geospatial Real-Time Aerial Image Display for a Low-Cost Autonomous Multispectral Remote Sensing Platform (AggieAir)

Austin M. Jensen
Utah State University

5-1-2009

gRAID: A Geospatial Real-Time Aerial Image Display for a Low-Cost Autonomous Multispectral Remote Sensing

Austin M. Jensen
Utah State University

gRAID: A GEOSPATIAL REAL-TIME AERIAL IMAGE DISPLAY FOR A LOW-COST AUTONOMOUS MULTISPECTRAL REMOTE SENSING PLATFORM (AGGIEAIR)

by

Austin M. Jensen

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. YangQuan Chen Major Professor	Dr. Wei Ren Committee Member
Dr. Rees Fullmer Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2009

Abstract

gRAID: A Geospatial Real-Time Aerial Image Display for a Low-Cost Autonomous Multispectral Remote Sensing Platform (AggieAir)

by

Austin M. Jensen, Master of Science

Utah State University, 2009

Major Professor: Dr. YangQuan Chen
Department: Electrical and Computer Engineering

Remote sensing helps many applications like precision irrigation, habitat mapping, and traffic monitoring. However, due to shortcomings of current remote sensing platforms - like high cost, low spatial, and temporal resolution - many applications do not have access to useful remote sensing data. A team at the Center for Self-Organizing and Intelligent Systems (CSOIS) together with the Utah Water Research Laboratory (UWRL) at Utah State University has been developing a new remote sensing platform to deal with these shortcomings in order to give more applications access to remote sensing data. This platform (AggieAir) is low cost, fully autonomous, easy to use, independent of a runway, has a fast turnover time, and a high spatial resolution. A program called the Geospatial Real-Time Aerial Image Display (gRAID) has also been developed to process the images taken from AggieAir. gRAID is able to correct the camera lens distortion, georeference, and display the images on a 3D globe, and export them in a conventional Geographic Information System (GIS) format for further processing. AggieAir and gRAID prove to be innovative and useful tools for remote sensing.

(80 pages)

Acknowledgments

I would like to thank all the members of CSOIS, especially Dr. Chen and the UAV team. First, thanks to Dr. Chen for giving me this opportunity, as well as financial support, advice, direction, and helping me understand what research is all about. Thank you Haiyang, Yiding, Cal, Dan, Di, Chris, and Mitch for your advice, support, criticisms, and late nights. Their hard work made this project a success and I could not have done it with out them. This also would not have been possible without the help of our additional German members of CSOIS, Marc Baumann and Daniel Kaplanek, who helped develop the imaging platforms and the image processing.

Thanks to the Utah Water Research Laboratory and Mac Mckee for the project inspiration and financial support. Thanks also to Shannon Clemens and Bushra Zaman for their remote sensing and GIS expertise.

I would also like to thank my committee members, Dr. Rees Fullmer and Dr. Wei Ren, for their willingness to support and provide their expertise throughout this project.

A great deal of gratitude also goes to my mom and dad. Thanks mom and dad for inspiring, encouraging, and teaching me how to work hard.

Above all, I would like to thank my wife, Katie, and my son Bryker for their sacrifice, love, and support through the countless hours and late nights. This would not have been possible without you.

Austin M. Jensen

Contents

List of Tables

List of Figures

Chapter 1
Introduction

1.1 Motivation

Optical remote sensing may be defined as the measurement of the reflectance or radiance of the electromagnetic spectrum from an object through an optical device. Examples of some optical devices include a camera, spectrometer, radiometer, and an antenna. Optical remote sensing is commonly used to measure the reflectance or radiance of the earth. Therefore, optical is normally omitted and referred to as just remote sensing. To obtain remote sensing data, one must be able to fly the optical device above the earth in order to get a good view of the area of interest (AOI). Currently, this is done using satellite or manned aircraft. The system carrying the optical instruments is often called a remote sensing platform. The data obtained from remote sensing, to directly or indirectly measure geoscientific variables, is used by many different applications: monitoring urban environmental quality, traffic monitoring, habitat mapping, and precision irrigation.

The environmental quality of a city is important to the people who live there [1], to people who are thinking about moving there and to government officials for city planning and improving the environment [2]. Pham and He [1] used imagery from the Quickbird satellite to study the relationship between green space types (e.g., parks, trees along a road, etc.) and the happiness of the city residents. The Quickbird satellite is a high resolution (2.8m) satellite with capabilities to provide imagery from the red, green, blue, and near-infrared (NIR) bands of the spectrum [3]. Through this study, Pham and He found a high correlation between the happiness of the residents and with their streets lined with trees. A negative correlation was found between resident happiness and agricultural areas. Yan et al. [2] also studied the environmental quality of a city by measuring the air quality, water quality, and the landscape degradation. While water quality can be measured directly with remote sensing, air quality is measured indirectly using the status of land use. Another factor in urban environmental quality is the urban heat island phenomenon. Urban heat island phenomenon is a modified, warmer thermal climate of an urbanized area when compared to the surrounding non-urbanized areas [4]. Murakami and Hoyano [5] monitored the urban heat island phenomenon by making air temperature maps using the canopy temperature of the trees. The canopy temperature was gathered using data from a remote sensing platform. This information would be important for establishing countermeasures against this phenomenon.

Traffic congestion is constantly increasing every day. While traffic increases, the tools to analyze and manage this traffic remains unable to fully understand it [6]. Palubinskas et al. [7] help to solve this problem using remote sensing. With an array of cameras, for a wider field of view (FOV), they were able to successfully model the traffic flow. Hoogendoorn et al. [8] argue that one can obtain a more accurate model by measuring the traffic information for each individual vehicle (microscopic level) as opposed to the flow of the highway (macroscopic level). To achieve this, Hoogendoorn et al. [8] use a manned helicopter with an imaging system to detect each individual vehicle. The properties extracted from the images include the vehicle heading, speed, and dimensions. Approximately 94% of the vehicles were sucessfully detected and tracked.

One of the most significant threats to global biodiversity and a functioning ecosystem is the spread of invasive plant species. To monitor the future expansion, the effectiveness of a control effort, or to identify targets for control activies (invasion fronts), one must know the spatial extent of the invasion to successfully manage the problem [9]. Many different types of invasive plant species exist today and are a threat to our ecosystem. Underwood et al. [9] used remote sensing to map Iceplant and Jubata along the coast of central California. Hyperspectral imagery was obtained with an airborne visible/infrared imaging spectrometer (AVIRIS) from a manned aircraft. Multispectral imagery from a satellite has also been proved effective for habitat mapping. Everitt et al. [3] used the red, green, blue, and NIR bands of the spectrum, from the Quickbird satellite, to detect giant reed infestations in a riparian area in southwest Texas. 95-100% accuracy was acheived.

An increasing population, resulting in industrial and agricultural water demands, has increased the competition over limited water resources. Since irrigation is the largest user of freshwater, a system to make crop management and irrigation more efficient could be used to help save water. Remote sensing could help make irrigation and crop management more efficient by proving a source of feedback to the farmer or to some automatic irrigation system (precision irrigation) [10]. Remote sensing can be used in precision irrigation (PI) by indirectly measuring the soil moisture or the evapotraspiration (ET). One way to measure the soil moisture is to measure the canopy temperature. Bajwa and Vories [11] measured the canopy temperature, the canopy reflectance and the soil moisture to see how well they correlated. By using the crop water stress index (CWSI) and the normalized difference vegetation index (NDVI), Bajwa and Vories showed that there is a high correlation between canopy temperature and reflectance with soil moisture tension. As shown by Hunsaker et al. [12], canopy reflectance can also be used to find ET. A NDVI-based crop coefficient method was used and proved to increase crop yield by 28%.

Many have shown that remote sensing can be a good tool for monitoring and measuring different geoscientific variables. However, current sources of remote sensing are too expensive, produce low-spatial resolution, have slow update rates, and a long turnover time. In precision irrigation, for example, Pinter et al. [13] comment that remote sensing would have to become more timely with high spatial and temporal resolution before it could become widely accepted by growers. To deal with these shortcomings, a low-cost, small, unmanned aerial vehicle (UAV) could be used as a remote sensing platform. In order to do this, here are some features a UAV platform would need to offer.

- Low-cost
- Easy-to-use
- Flexible
- High update rate

- Quick turnover time
- High resolution

To allow users from many backgrounds to use the platform, it should be low-cost and easy-to-use. This would require almost complete autonomy with system deployment, system operation, and image processing. It should also be flexible with the ability to operate from almost anywhere at nearly anytime, given appropriate weather conditions. To operate the platform at almost any location, the independence of a runway is very important. Also if the platform was able to deploy at anytime, the data could be updated at a high rate. However, this would require a fast turnover time which would also necessitate near-autonomous image processing. High resolution and accurate georeferencing would also be important for high quality data.

1.2 Literature Review

Most research motivation concerning UAVs has been military related. This includes using UAVs for target detection, localization, and tracking. Some non-military applications for UAVs have also been studied in the area of remote sensing and ecological applications. Most of the UAV platforms used for military and non-military applications has either been conducted with fixed wing aircraft (airplanes, flying wings) or rotary wing aircraft (helicopters).

Rotary wing aircraft has been demonstrated to be a good platform for remote sensing. For example, Berni et al. [14] investigated the use of a thermal camera and a 6-band multispectral visual camera, mounted on a helicopter, to detect water stress and to measure the biophysical parameters of crops. The study was successful showing that both the thermal camera and the multispectral camera were able to detect water stress. In addition, the multispectral camera was able to distinguish biophysical parameters of crops. Berni et al. [14] mention that using helicopters as remote sensing platforms proved to be useful for these types of experiments. However due to the short endurance and limited range, it may not be the best platform for practical applications. Eisenbeiss et al. [15] also used an autonomous helicopter for remote sensing. Equipped with a charge-coupled device (CCD) camera, along with data from a laser scanner, the helicopter was utilized to create a 3D model of an archaeological site in Peru. The remote location and rugged topography usually makes getting to and working at the site difficult, however the small UAV proved to be a good tool to offset these drawbacks.

For longer flight time, faster speed, and longer range, others have used fixed wing UAVs for remote sensing. For example, Johnson et al. [16] used a large, solar-powered, NASA UAV called Pathfinder to detect the ripeness of a coffee field. Furthermore, Johnson et al. [17] used a smaller UAV called RCATS/APV-3 to investigate the feasibility of using a UAV to support agricultural monitoring needs. Both of these studies were successful at showing that a UAV has the potential to make a great contribution to the remote sensing community. However, the UAVs Johnson et al. [16, 17] used were very large and expensive. In another application, Graml and Wigley [18] proposed a method to detect hotspots around the perimeter of a recently extinguished bushfire using a small IntelliTech Microsystems Vector-P UAV. By detecting these hotspots, this method can help firefighters to prevent the bushfire from reigniting. Using UAVs for firefighting is especially helpful because UAVs are able to enter hazardous areas without endangering human life. Casbeer et al. [19, 20] also proposed a method to help firefighters during a forest fire. In this method, Casbeer et al. [19, 20] used multiple UAVs to detect the boundary of the forest fire. Simulations showed that the latency between boarder update rates can be minimized using consensus between each UAV.

All the advantages of small, low-cost, UAVs are increased when using multiple UAVs (a coven) to achieve one goal. These advantages are shown through research conducted by Drake et al. [21] who used a coven to detect the location of a radar unit for military use. Virtual vector fields were used to hold good geometry between aircraft, avoid collision, avoid no fly zones, and circle the radar while localizing it. The location of the radar is calculated by comparing the time difference the radar signal reaches the different members of the coven. This triangulation method is called time difference of arrival (TDOA). Because the TDOA must be measured concurrently from different locations, successful and accurate radar detection may not be possible without more than one UAV. Pachter et al. [22] also used the advantages of a coven to achieve better localization accuracy of a target. While flying two different planes independently, each plane measured the location of the target with an error of about 10m. Using both aircraft together, the localization error was decreased to 5m. In an ecological application, Chao et al. [23] talked about using a coven for band-reconfigurable cooperative remote sensing. A small, single UAV may not be able to carry enough cameras to cover all the necessary bands of the spectrum needed for an application. However if a coven of UAVs is exercised, and each UAV carries its own band of the spectrum, this may be feasible. Furthermore, the coven may be reconfigured for different AOIs which require different bands of the spectrum.

1.3 Contributions

In order to make remote sensing data more available, a team at the Center for Self-Organizing and Intelligent Systems (CSOIS) has been developing a remote sensing platform (AggieAir) that has all the following features.

- Fixed wing
- Low cost
- Small
- Fully autonomous
- Easy-to-use
- Independent of runway
- Coven capable
- High spatial resolution
- Fast turnover

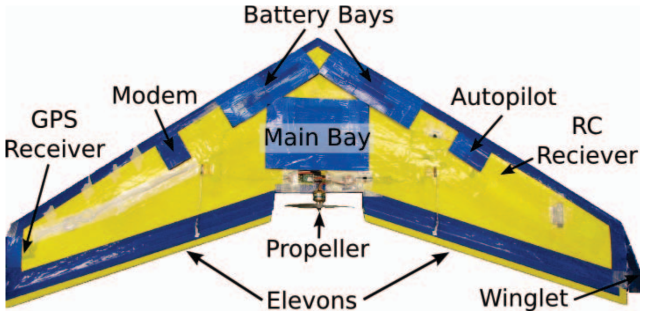
An element within the AggieAir system is a software package capable of orthorectifying, processing, and displaying the images in real-time or post flight. This software package, called Geospatial Real-Time Aerial Image Display (gRAID), is the contribution detailed in this thesis.

1.4 Organization

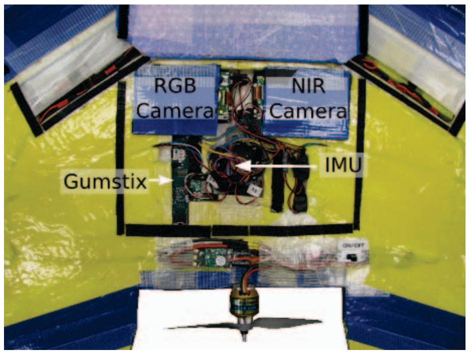
An introduction of AggieAir will first be presented including details on the autopilot and the imaging systems developed for the platform. Then details about how to use gRIAD and what all the different tools do will be introduced. This will be followed by image processing details including, camera calibration, radial distortion, color converting georeferencing, drawing the images on World Wind, and making world files and concluded with a summary of the contributions and suggestions for future work.

Chapter 2
AggieAir: A Low-Cost Multispectral Remote Sensing Platform

AggieAir is an autonomous, multispectral remote sensing platform. The system only needs two people to operate and is not dependent on a runway. It is launched using a Bungee and glides to the ground for a skid landing. Figure 2.1 shows the layout of AggieAir's components. In the front of the aircraft, the battery bays hold eight 12v batteries. This supplies enough power to the aircraft to keep it in the air for approximately an hour. The main bay comprises of two cameras and the inertial measurement unit (IMU). One camera takes pictures in the red, green, and blue bands of the spectrum and the other takes pictures in the near infrared band (NIR) of the spectrum. The IMU and the GPS receiver are the aircraft sensors. These sensors measure the orientation and the position of the aircraft respectively. Data from the IMU and the GPS module are processed and sent to the autopilot by the Gumstix. The Gumstix is an on-board computer which not only sends data to the autopilot, but also controls the cameras. While receiving position and orientation data from Gumstix, the autopilot navigates the aircraft according to a preprogrammed flight plan. The autopilot uses the elevons and the propeller to navigate the aircraft. The elevons are used to rotate the aircraft and the propeller to accelerate the aircraft. The winglets also help navigate the aircraft by adding stability around the z axis. The position, orientation, aircraft status, and other variables (telemetry) are transmitted through a wireless modem down to the ground control station (GCS). Using this data, the user at the GCS can watch the aircraft and give it commands. The user can also take manual control of the aircraft using a radio-controlled (RC) transmitter. Different controls on the transmitter send different commands to the aircraft via the RC receiver. The RC receiver interprets the commands from the transmitter and moves the elevons and propeller



(a) Aircraft Component Layout



(b) Main Bay Component Layout

Fig. 2.1: AggieAir layout.

accordingly. In addition, the pilot can use the RC transmitter to switch from autonomous flight to manual at any time. Figure 2.2 shows a block diagram to help illustrate how each component of AggieAir works collectively.

2.1 Paparazzi

Paparazzi [24] is an open source autopilot developed by a community of people from all over the world. Due to Paparazzi being open source, the aircraft code, ground station code, and hardware schematics are all available to the public. The availability of this information makes it possible to adapt Paparazzi for specific applications. For example, Paparazzi usually comes standard with infrared sensors to measure the orientation of the aircraft. This method works fine for navigating an aircraft, however the level of accuracy is not good enough for georeferencing aerial images. Since the code is open source, CSOIS adapted Paparazzi to use a more accurate orientation sensor: an IMU.

2.1.1 Paparazzi Center

Paparazzi Center is the backbone of Paparazzi. This application is employed to setup, compile and program the UAV, and execute additional processes (agents) used to monitor, simulate, replay, and plot telemetry data from the UAV. Figure 2.3 shows the layout of Paparazzi Center.

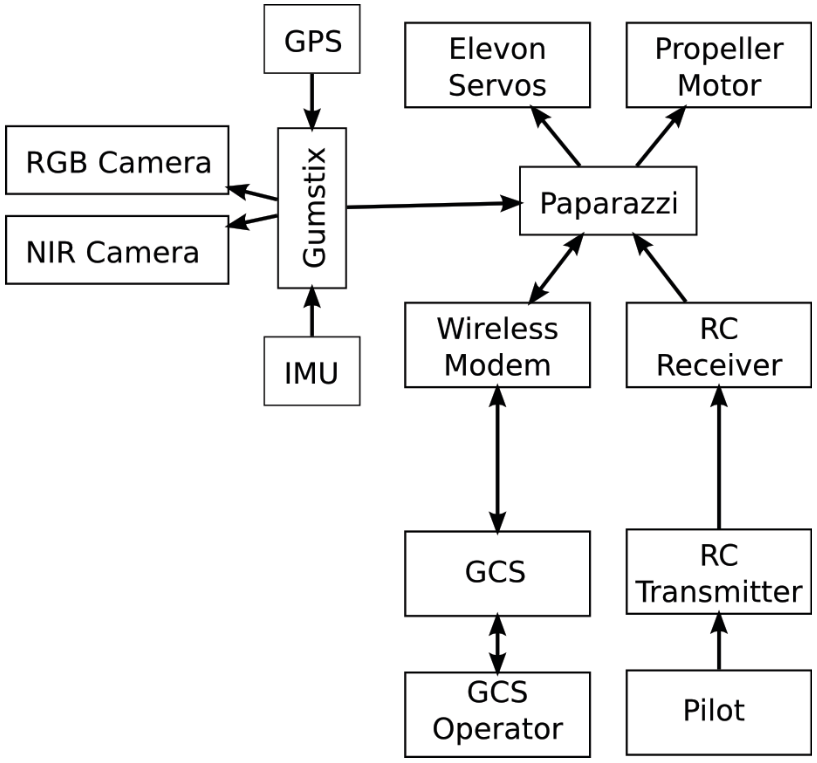


Fig. 2.2: AggieAir flow diagram.

The aircraft configuration section of Paparazzi Center allows users to manage the airborne code and the various settings for the UAV. Each aircraft configuration is defined by five configuration files: the airframe, flight plan, settings, radio, and telemetry configuration files. Each aircraft configuration is also defined by an aircraft name, number, and color. These properties are important for identifying each individual aircraft while flying multiple UAVs at the same time. Each configuration can be saved and selected using the A/C combo box in the top-right corner.

Once the aircraft configuration is set, the building section of Paparazzi Center is used to construct the code defined by the configuration files. The code can be built either for the airborne code on the aircraft or for a simulation. If the code was built for the aircraft, the upload button will program the aircraft through a USB interface with the aircraft. If the code was built for simulation, the simulator may be ran using the controls from the execution section.

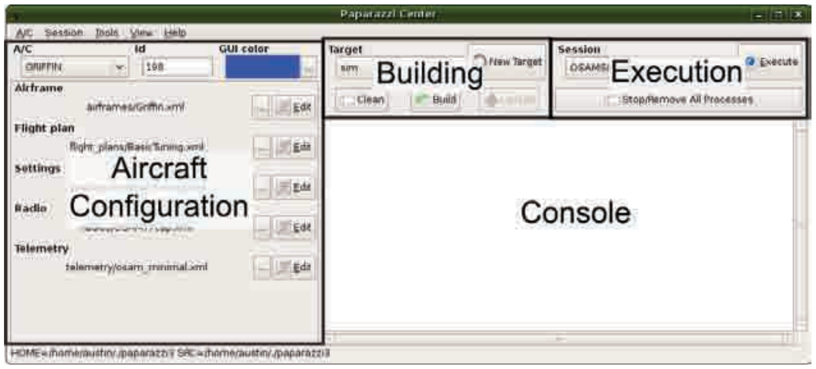


Fig. 2.3: Paparazzi center.

The execution section is used to manage Paparazzi agents. Agents are individual processes with a specific task. Usually one agent is useless alone and must work with a group of agents to make something meaningful. These groups are organized into sessions. A session is saved with a list of agents and an optional set of arguments for each agent. When the session is implemented, it executes all the agents with their saved arguments. Sessions are used, for example, to monitor the aircraft, simulate a flight plan, or replaying a datalog. The agents which are running are displayed below the execution and build sections of Paparazzi Center.

Agents share data through a decentralized software bus called Ivy. It allows the agents to exchange data with the illusion of broadcasting the information. Using the series of multiplatform, multilanguage Ivy libraries, the agents can connect to the bus and start listening and or broadcasting.

The console is used to report errors while building the code and while the code is running (on the aircraft or during simulation).

2.1.2 Aircraft Configuration Files

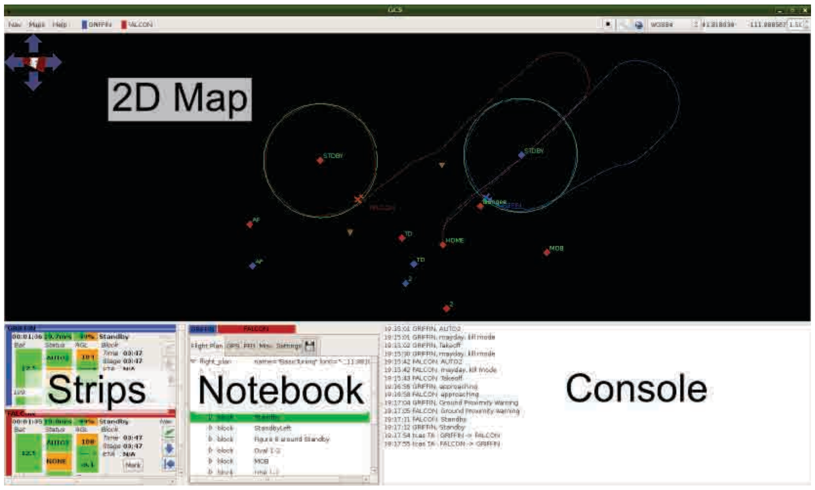
The airframe configuration file defines variables dealing with flight performance like controller gains, trim values, and physical constraints. These are variables which would be unique for each airframe and must be tuned. The flight plan contains waypoints and blocks which are used to tell the aircraft where to go. A waypoint is a point of interest on the map defined by its location (GPS and altitude). The blocks use the waypoints to give specific commands to the aircraft. An example of a block is the Goto Block. The Goto Block simply tells the aircraft to go to a given waypoint. Another block is the Circle Block, which tells the aircraft to circle around a given waypoint at a given radius. Exceptions can also be used in the flight plan to detect specific conditions and to redirect the aircraft accordingly. For example, an exception could be used to go to a waypoint if the plane gets too far away from home. More advanced navigation routines can also be included in the flight plan (e.g., takeoff and landing). More details of advanced navigation routines can be found in appendix B. The settings configuration file contains information on special variables which can be updated, from the GCS, while the UAV is in the air. The radio configuration file configures the UAV so it is compatible with the RC transmitter, and the telemetry configuration file declares the various variables sent down from the UAV to the GCS.

2.1.3 The Ground Control Station (GCS) and Other Paparazzi Agents

The Paparazzi GCS is one of the most important Paparazzi agents. It is used to monitor and control the UAV while in flight or in simulation. Figure 2.4 shows the layout of the GCS.

The 2D map gives the user an aerial perspective to help control and monitor the aircraft. The aircraft, the waypoints, the path of the aircraft, and the desired path of the aircraft are all displayed on the 2D map. To help know where the aircraft is, background images can be downloaded from Google maps under the Maps menu. The 2D map can be navigated using the mouse, the arrow keys, or by using the menus and buttons above the map.

Each strip on the GCS displays important telemetry data and has buttons for common commands for the respective aircraft. Each visible aircraft to the GCS will have its own strip. Examples of the telemetry data displayed on each strip include battery voltage, speed, throttle, current altitude, target altitude, and the autonomous mode. In addition to common command buttons (e.g., launch, kill throttle, altitude, and lateral shift), it has the



```
19:35:01 GRIFFIN, AUTO2
19:15:01 GRIFFIN, mayday, kill mode
19:15:02 GRIFFIN, Takeoff
19:15:30 GRIFFIN, mayday, kill mode
19:15:42 FALCON, AUTO2
19:15:42 FALCON, mayday, kill mode
19:15:43 FALCON, Takeoff
19:16:56 GRIFFIN, approaching
19:16:58 FALCON, approaching
19:17:04 GRIFFIN, Ground Proximity Warning
19:17:05 FALCON, Ground Proximity Warning
19:17:11 FALCON, Standby
19:17:12 GRIFFIN, Standby
19:17:54 tcas TA : GRIFFIN -> FALCON
19:17:55 tcas TA : FALCON -> GRIFFIN
```

Fig. 2.4: Paparazzi ground control station.

option to add more buttons which represent different blocks in the flight plan.

Like the strips, the notebook frame contains a page for each running aircraft. Each page has multiple subpages which contain tools for monitoring and controlling the aircraft. The flight plan subpage is used to display all the elements in the flight plan. It also allows the user to change the current block being executed (highlighted in green). The GPS, PFD, and Misc subpages all display information about the UAV. The GPS displays the number of satellites and the position error of the GPS signal, the PFD displays the orientation of the aircraft, and the Misc subpage displays other information like wind information. The settings subpage contains all the settings of the UAV which the user can change during flight. These settings include the controller gains from the airframe configuration file, the kill throttle, and other flight parameters. Because of the access to the controller gains, the settings subpage is important for tuning the aircraft.

The console frame displays messages and alerts the user when the status of the aircraft has changed.

2.2 Imaging Systems

For AggieAir to become a good remote sensing platform, the imaging system is very important. In order to be successful in areas like habitat mapping and precision irrigation, not only should it have the ability to detect light in the red, green, and blue bands of the spectrum, but also the NIR band. Fortunately, there is a simple solution for converting a conventional camera into a NIR camera. All CCD cameras are sensitive from 400 to 1200nm. Camera manufactures install NIR filters in order to prevent the NIR light from distorting the image quality. By removing this filter, the camera is then capable of detecting a band of the spectrum comparable to the Landsat or Quickbird satellites by installing other appropriate filters. Furthermore, multiple cameras at different bands of the spectrum could represent a multispectral remote sensing platform. A few different imaging systems have been developed with these capabilities for AggieAir.

2.2.1 GhostFinger (GF)

GhostFinger (GF) [25] is the first imaging system built for AggieAir. It uses some additional hardware interfaced to a Pentax camera to physically trigger the camera to take a picture (fig. 2.5). Although this method proved effective for periodically taking pictures from the UAV, the position and orientation of the camera when the image was taken was unknown. This presented a problem when trying to georeference the image. In order to synchronize the image with the position and orientation of the UAV, a new version of GF was developed. This new version included a GPS module, an inclinometer, a pressure sensor, and an SD card. The GPS module and the pressure sensor were used to determine the position of the UAV. The inclinometer was used to determine the orientation of the UAV. And the SD card stored the position and orientation data every time a picture was taken. Everything worked well with this system except for the inclinometer. It was not built for dynamic motion and did not report correct orientation values from the aircraft. Refer to table 2.1 for more information on GhostFinger.

2.2.2 GhostFinger Video (GF-Video)

Another AggieAir imaging system is called GhostFinger Video (GF-Video). In this system (fig 2.6), a video stream from a small video camera is transmitted to the ground station where it is captured and synchronized with the position and orientation of the UAV.

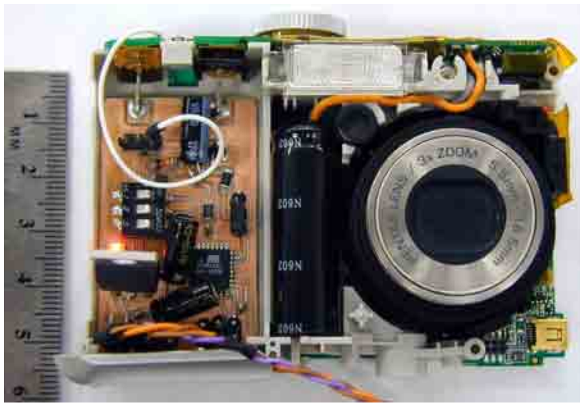


Fig. 2.5: GhostFinger.

Table 2.1: GhostFinger specifications (200m height).

Resolution (pixels)	3072x2304
Focal Length (mm)	6
Field of View (deg)	50x39
Ground Resolution (m)	0.09
Swath Width (m)	276
Weight (g)	200
Frequency (FPS)	0.3

The synchronization was done by connecting the computer, which captured the images, with the GCS and reading the data off the Ivy bus. Even though the video cameras have a lower resolution, the images can be georeferenced and viewed on a map in real-time. This allows the user to check and confirm the AOI was covered before the plane lands. Transmitting the images to the ground station is also nice for finding and tracking targets in real-time. Table 2.2 contains more information on the GF-Video system.

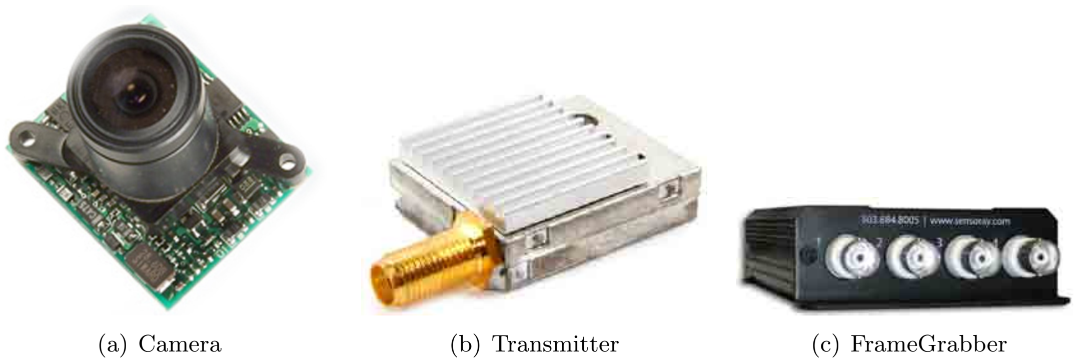

(a) Camera (b) Transmitter (c) FrameGrabber

Fig. 2.6: GhostFinger Video.

Table 2.2: GhostFinger video specifications (200m height).

Resolution (pixels)	640x480
Focal Length (mm)	1, 3, 8
Field of View (deg)	97x80, 60x45, 24x19
Ground Resolution (m)	0.70, 0.35, 0.13
Swath Width (m)	454, 230, 86
Weight (g)	110
Frequency (FPS)	30

2.2.3 GhostFoto (GFoto)

The imaging system currently being used for AggieAir is called GhostFoto (GFoto). It borrows the idea of controlling a conventional point-and-click camera with some external hardware from GF. The only difference is that the external hardware controls the camera through a software interface instead of hardware. GFoto uses a Gumstix embedded computer with an open source library called gphoto to control a Canon Powershot 100sx camera.

Figure 2.1(b) shows the GFoto system hooked up together and fig. 2.7 shows the Canon camera after the cover has been removed. This interface makes it easier to synchronize position and orientation data with the images because the Gumstix is also connected to the IMU and GPS module used to navigate the aircraft. The software interface also makes real-time, on board image processing possible because the Gumstix is actually able to take, process, and store the images from the camera. Another improvement GFoto provides is the image quality. The Pentax camera did not allow the user to set all the optical settings manually. Settings like the exposure time could be set manually however other settings like the white balance could not. While flying, the auto-white balance changes the brightness of the camera from picture-to-picture and makes a non-uniform map. The Canon camera allows the user to manually set all of the settings. Furthermore, these settings can be set with the gphoto library on the Gumstix. One drawback to GFoto is that it is the heaviest system and takes up the most space. Table 2.3 contains more information on the GFoto system.

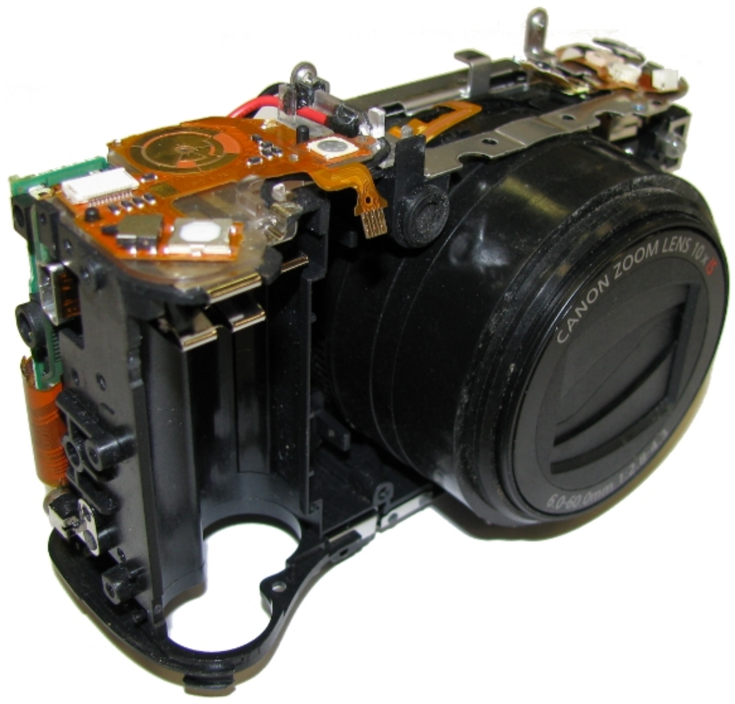


Fig. 2.7: The Canon camera used for GFoto.

Table 2.3: GhostFoto specifications (200m height).

Resolution (pixels)	3264x2448
Focal Length (mm)	6
Field of View (deg)	50x39
Ground Resolution (m)	0.05
Swath Width (m)	190
Weight (g)	250
Frequency (FPS)	0.3

Chapter 3
Geospatial Real-Time Aerial Image Display (gRAID)

The Geospatial Real-Time Aerial Image Display (gRAID) is a plug-in for a 3D interactive world viewer called World Wind [26]. gRAID takes the images from the aircraft, as well as its position and orientation, and overlays them on the 3D earth generated by World Wind. Furthermore, this process can be done post-flight or in real-time while the plane is flying. gRAID also has the capabilities to correct the radial distortion in the images and to create a gray scale image from a single RGB channel. After the images are uploaded on World Wind, many options can be performed. For example, the images can be organized into different datasets, deleted, and moved around the map by changing the respective aircraft data. Furthermore, the images can be converted into world files and loaded into conventional GIS software for further, advanced image processing.

3.1 World Wind

World Wind (fig. 3.1) was created by NASA, released in 2004 and is now developed by NASA staff and an open source community. It allows the user to rotate the globe, zoom to any altitude, and rotate the camera to view the earth any angle. To enhance the realistic view, World Wind comes with satellite imagery from Landsat and Shuttle Radar Topography Mission terrain data.

World Wind was chosen as a foundation for gRAID foremost to access the open source code as well as the many tools and plug-ins which are already created for World Wind. One useful plug-in is Virtual Earth. The Virtual Earth plug-in downloads high resolution images from Microsoft's local.live.com and overlays them on the earth. Other plug-ins, like the boarder and placenames, help navigate the globe by displaying state and country boarders, as well as the names of cities, countries, regions, and other areas of interest.

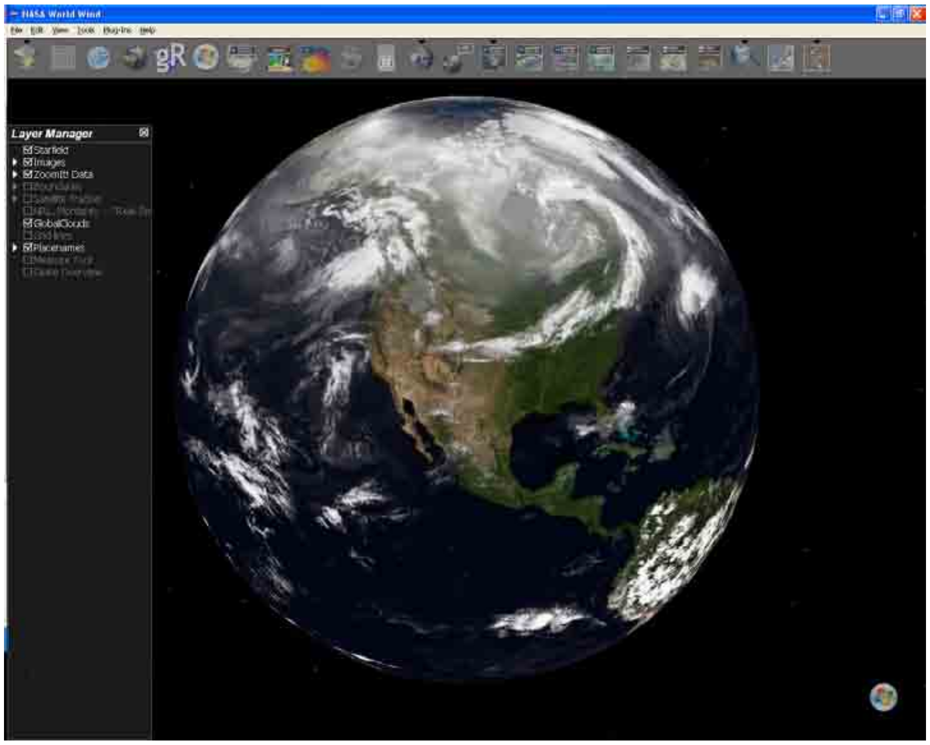
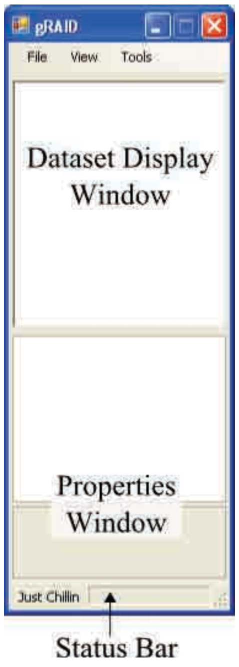


Fig. 3.1: World Wind screen shot.

In addition, the measure tool can be used to measure distances in the images. Since a community of people contribute to World Wind, new plug-ins are constantly being created. This type of development is very useful because one can prevent recreating something that has already been created. For more details of how to create plug-ins for World Wind, refer to appendix A.

3.2 The gRAID Form

To use gRAID, click on the gRAID button in the toolbar (fig. 3.2(b)). The gRIAD button should be in the toolbar as long as gRAID is loaded into World Wind [26]. The gRAID Form (fig. 3.2(a)) will appear in the left side of the World Window over the Layer Manager. The gRAID Form manages the current dataset and launches tools to manipulate the images in the dataset.



(a) gRAID Form

(b) gRAID Button

Fig. 3.2: Starting gRAID.

3.3 The gRAID Dataset

Each dataset contains a collection of images. In addition, datasets can also contain other datasets. Datasets help organize the images in order to help the user process the data more efficiently. A new dataset can be created my selecting the "new" menu item in the main menu.

Figure 3.3 shows the Dataset Display Window with a new dataset and the Dataset Display Menu. This menu can be accessed using the right side button on the mouse. The items inside the dataset display menu change depending on the selected item. Through this menu, the user could add another dataset or import images in the selected dataset. Many different images can be imported into a dataset. One type of image is a world file which has already been georeferenced and is the easiest to import. A world file actually consists of two files: an image and a text file. The image is an aerial photo which has already been corrected and georeferenced. In addition, the top edge of the image runs parallel with the east-west direction and points north. The text file associated with a world file contains the

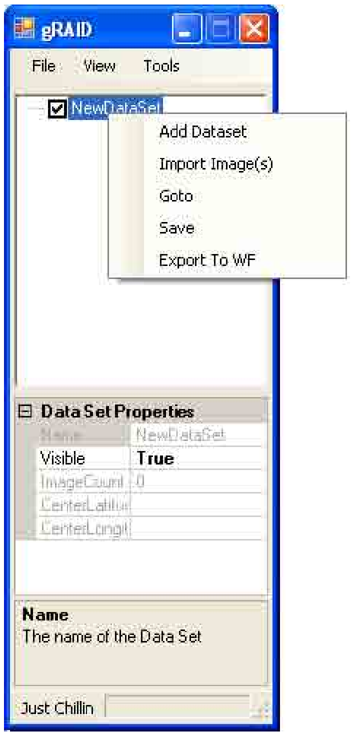


Fig. 3.3: New dataset with menu.

position of the top right corner of the image and the distance of each pixel in the image. The extension of the world file consists of the first and last letters of the image extension followed by a "w." For example, a .jpg image extension would have a corresponding .jgw text file. To import a world file into a dataset, select the Import Images(s) item in the Dataset Display Menu. A dialog will appear allowing the user to select a group of images. If the images correspond to world files, gRAID will add them to the dataset and display them on the globe. Other wise, the images will be added to the dataset without any corresponding aircraft data and will not be displayed on the globe. The images, without aircraft data, can still be georeferenced and displayed on World Wind, however the user must manually enter the aircraft data.

After the images are added to a dataset, they will appear below it in the Dataset Display Window. Figure 3.4 shows the result of adding two datasets (UpperRoad and LowerRoad) within the WorldFile dataset. World file images are then added to UpperRoad and LowerRoad. The names of the datasets are changed by clicking on the dataset twice and

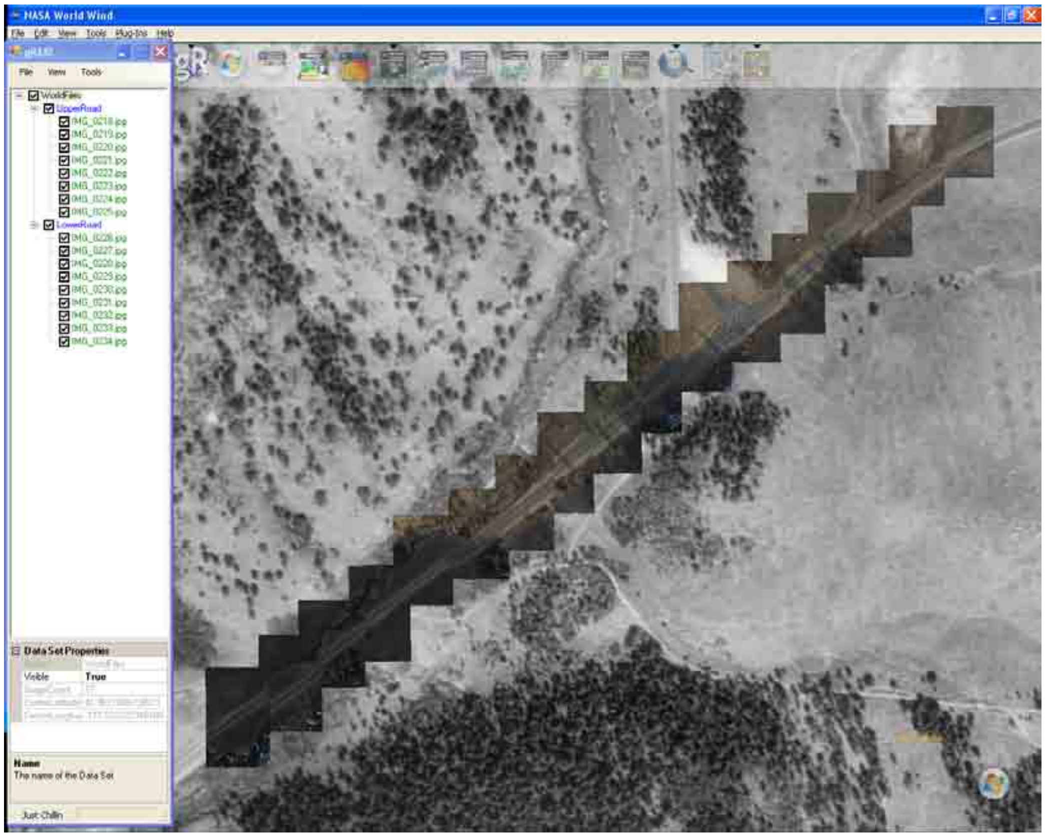


Fig. 3.4: gRAID with world files.

entering in the desired name. In this list, datasets and images can easily be distinguished from each other by their different colors; datasets are blue and images are green. The check boxes next to the items allow the user to hide or show the images on the globe. If a dataset is unchecked, all the images inside that dataset will hide; checking the dataset will show all the images on the globe. This operation is recursive and will also effect the datasets inside the hidden dataset.

The Dataset Display Menu also contains a Goto menu item (fig. 3.3). This command helps the user navigate by displaying the selected item on the globe within the field of view of the world window. For example, if a dataset is selected, the camera will go to a point over the globe so all the images within that dataset are shown.

There are two different ways to save a dataset. The top dataset (e.g., WorldFile dataset in fig. 3.4) can be saved using the Save menu item under the File main menu. When a dataset is saved, all the information collected during the import is saved in a gRAID dataset file (.gds). This makes it faster to load a dataset file than to import the images again. Furthermore, the organization and structure is maintained. The dataset file can be reloaded using the load menu item under the File main menu. Each dataset within the top dataset can also be saved in a separate dataset file by selecting the Save menu item in the Dataset Display Menu. This dataset can also be loaded into gRAID as the top dataset. To close the current top dataset, it can be closed using the Close menu item under the File main menu.

Datasets and images can also be deleted from the list. This can be done using the delete menu item in the Dataset Display Menu or by using the delete key on the keyboard. The only dataset that cannot be deleted is the top dataset.

3.3.1 Images in the World Window

Georeferenced images in a dataset are shown on the World Window. When an image is imported into a dataset, multiple layers of the image are created, each with a different resolution. Most of the images displayed on the globe are the lowest layer and have the lowest resolution. However, the high resolution top layer image is displayed when the mouse pointer is positioned over the image or if the image is selected in the Dataset Display Window. Other features which appear if the image is selected include, a black box around the image, displaying the name over the image, and the image is drawn over all the other images around it (fig. 3.5). The user can also hide and delete images in the World Window by right clicking the mouse over the image and selecting the appropriate item in the popup menu.

3.3.2 Types of Images

The different types of images which can be imported into a dataset including world files, images without aircraft data, and aircraft images. World files and images without aircraft data have already been explained. World files are images which are corrected and georeferenced; images without aircraft data are aerial photos without any information

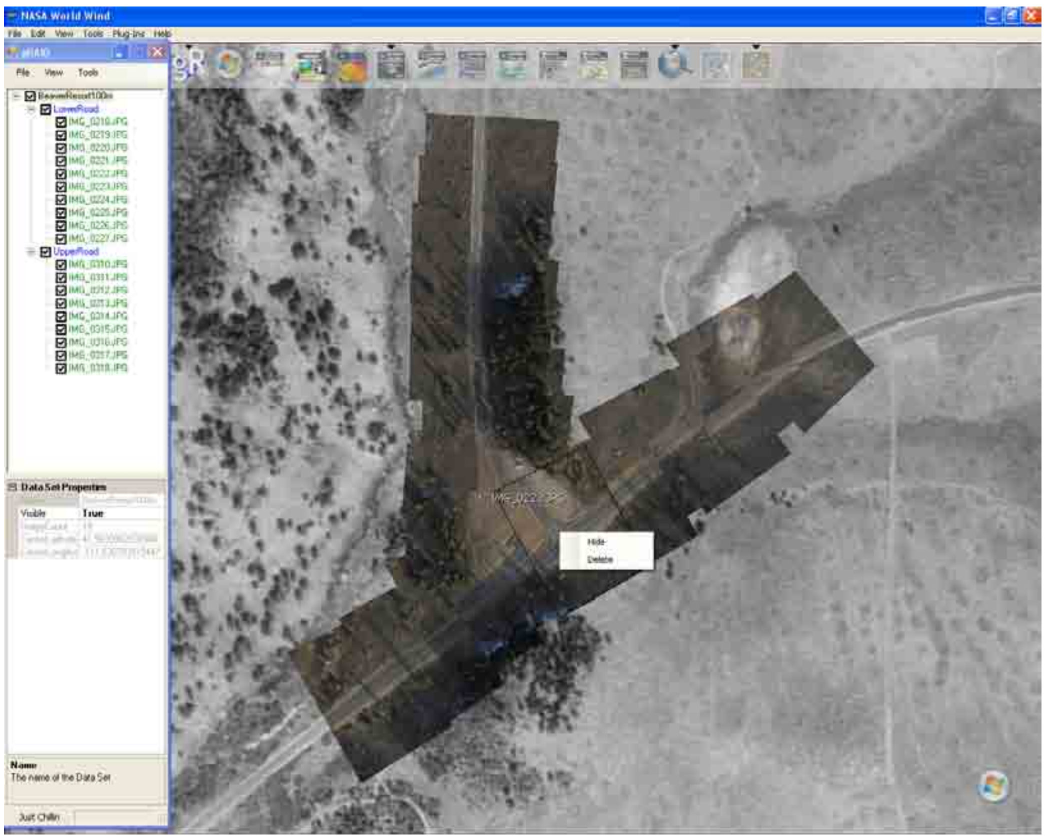


Fig. 3.5: Selecting images in the World Window.

pertaining to where they may be located on the earth. However, the images without aircraft data can be georeferenced by providing the position and orientation of the aircraft when the image was taken. When this information is provided, the images without aircraft data are converted into aircraft images and displayed on the World Window.

Using the gRAID Datalog (GDL) Dialog (fig. 3.6), aircraft images can be imported without having to manually specify the aircraft data for each image. To import aircraft images directly, select the Import Image(s) menu item in the Dataset Display Menu. Instead of selecting the images to import, select a GDL file. When a GDL file is selected, the GDL Dialog is displayed and contains all the images within the GDL file. From the GDL dialog, the user can select the camera settings, grouping options, and the image channels to include in the images. The grouping options let the user organize the images into different sizes of groups. For example, the images displayed in the GDL Dialog could be organized into datasets of 10, 15 or 20. Grouping the images into smaller sizes makes them more manageable. The images displayed in the GDL Dialog can be filtered using the filter section of the GDL Dialog. The user can specify the tolerable limits of roll, pitch and altitude in addition to the frequency of images to use. Once the refresh button is pushed, the images displayed by the GDL Dialog are updated with the filter values. Images can also be manually removed using the delete button and the shift+delete button on the key board. The delete button will delete the selected images. The shift+delete button will delete all the images besides the selected ones. Once the Ok button is pressed, the images displayed in the GDL Dialog are added to the selected dataset. Figure 3.5 shows aircraft images in the World Window.

3.3.3 Camera Objects

To correctly and accurately georeference the images from the aircraft, it is important to know the properties of the camera. gRIAD allows the user to store multiple cameras and their properties in the gRAID Object Dialog (fig. 3.7(a)). These cameras can be used later while importing images into a dataset. To add a camera to gRAID, click on the small button on the right side of the gRIAD Object Dialog (with the three dots in

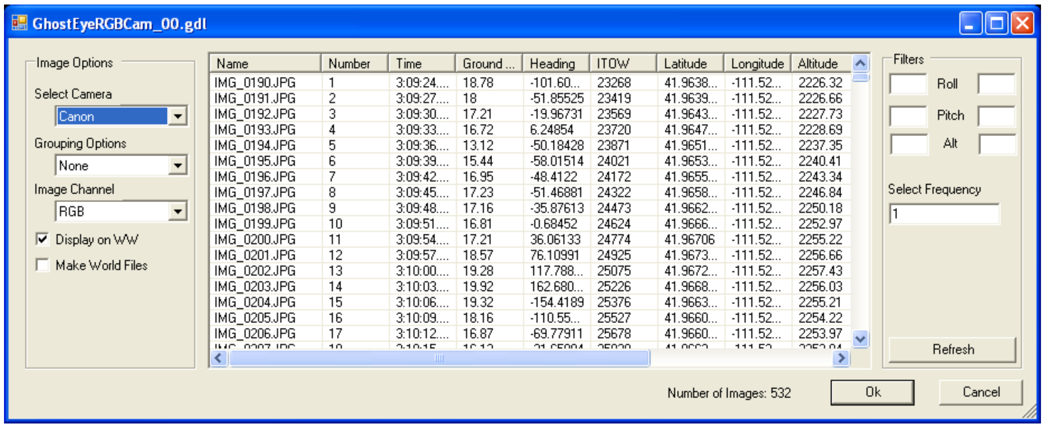

GhostEyeRGBCam_00.gdl

Image Options

Select Camera: Canon

Grouping Options: None

Image Channel: RGB

☑ Display on WW

☐ Make World Files

Name	Number	Time	Ground ...	Heading	ITOW	Latitude	Longitude	Altitude
IMG_0190.JPG	1	3:09:24....	18.78	-101.60...	23268	41.9638...	-111.52...	2226.32
IMG_0191.JPG	2	3:09:27....	18	-51.85525	23419	41.9639...	-111.52...	2226.66
IMG_0192.JPG	3	3:09:30....	17.21	-19.96731	23569	41.9643...	-111.52...	2227.73
IMG_0193.JPG	4	3:09:33....	16.72	6.24854	23720	41.9647...	-111.52...	2228.69
IMG_0194.JPG	5	3:09:36....	13.12	-50.18428	23871	41.9651...	-111.52...	2237.35
IMG_0195.JPG	6	3:09:39....	15.44	-58.01514	24021	41.9653...	-111.52...	2240.41
IMG_0196.JPG	7	3:09:42....	16.95	-48.4122	24172	41.9655...	-111.52...	2243.34
IMG_0197.JPG	8	3:09:45....	17.23	-51.46881	24322	41.9658...	-111.52...	2246.84
IMG_0198.JPG	9	3:09:48....	17.16	-35.87613	24473	41.9662...	-111.52...	2250.18
IMG_0199.JPG	10	3:09:51....	16.81	-0.68452	24624	41.9666...	-111.52...	2252.97
IMG_0200.JPG	11	3:09:54....	17.21	36.06133	24774	41.96706	-111.52...	2255.22
IMG_0201.JPG	12	3:09:57....	18.57	76.10991	24925	41.9673...	-111.52...	2256.66
IMG_0202.JPG	13	3:10:00....	19.28	117.788...	25075	41.9672...	-111.52...	2257.43
IMG_0203.JPG	14	3:10:03....	19.92	162.680...	25226	41.9668...	-111.52...	2256.03
IMG_0204.JPG	15	3:10:06....	19.32	-154.4189	25376	41.9663...	-111.52...	2255.21
IMG_0205.JPG	16	3:10:09....	18.16	-110.55...	25527	41.9660...	-111.52...	2254.22
IMG_0206.JPG	17	3:10:12....	16.87	-69.77911	25678	41.9660...	-111.52...	2253.97

Filters: Roll, Pitch, Alt

Select Frequency: 1

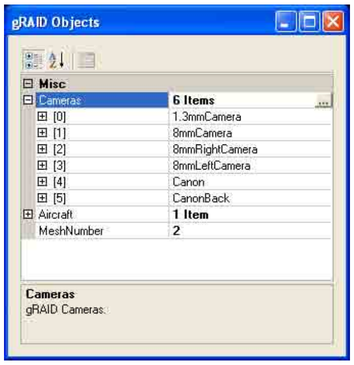
Refresh

Number of Images: 532

Ok Cancel

Fig. 3.6: The gRAID Datalog Dialog.

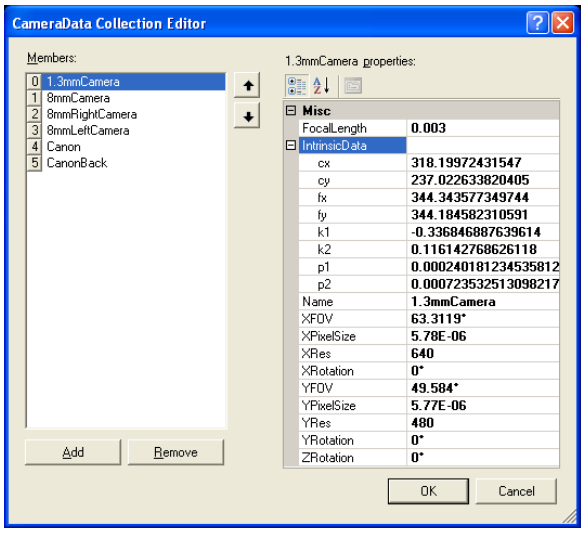
it). This brings up another dialog called the Camera Properties Dialog (fig. 3.7(b)). The Camera Properties Dialog gives more details about each camera object and allows the user to create new cameras and remove existing cameras. All the properties listed in the Camera Properties Dialog are required except for the intrinsic data. Most of the data (e.g., focal length, FOV, etc.) is provided by the manufacture and is sufficient to georeference the images. However, calibrating the cameras and obtaining the camera's intrinsic data can improve the accuracy of the orthorectification and is recommended.


gRAID Objects

Misc	
Cameras	6 Items
[0]	1.3mmCamera
[1]	8mmCamera
[2]	8mmRightCamera
[3]	8mmLeftCamera
[4]	Canon
[5]	CanonBack
Aircraft	1 Item
MeshNumber	2

Cameras
gRAID Cameras.

(a) gRAID Object Dialog With Cameras


CameraData Collection Editor

Members:

0 1.3mmCamera
1 8mmCamera
2 8mmRightCamera
3 8mmLeftCamera
4 Canon
5 CanonBack

Add Remove

1.3mmCamera properties:

Misc	
FocalLength	0.003
IntrinsicData	
cx	318.19972431547
cy	237.022633820405
fx	344.343577349744
fy	344.184582310591
k1	-0.336846887639614
k2	0.116142768626118
p1	0.000240181234535812
p2	0.000723532513098217
Name	1.3mmCamera
XFOV	63.3119°
XPixelSize	5.78E-06
XRes	640
XRotation	0°
YFOV	49.584°
YPixelSize	5.77E-06
YRes	480
YRotation	0°
ZRotation	0°

OK Cancel

(b) Camera Properties Dialog

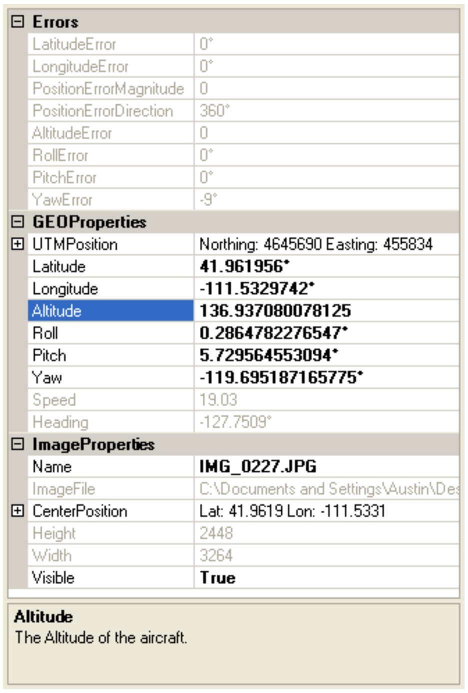
Fig. 3.7: gRAID camera object windows.

3.3.4 Property Window

Properties from each item in the Dataset Display Window are displayed in the Properties Window (fig. 3.2(a)). Figure 3.8 shows the different types of items in the Dataset Display Window and their different properties. Not only does the Property Window display valuable information to the user, but it also allows the user to change the properties in bold. The section at the bottom of the property window displays information about the selected property.

The dataset has five properties (fig. 3.8(c)): dataset name, whether or not the dataset is visible, how many images are in the dataset, and the position of the center of the dataset. The visible property operates in the same way as the check boxes in the Dataset Display Window. If the visible property is false, the dataset is not displayed on the globe; if the property is true, the dataset is displayed. The center position of the dataset is used when the user clicks the Goto menu item. This is the position the camera moves to.

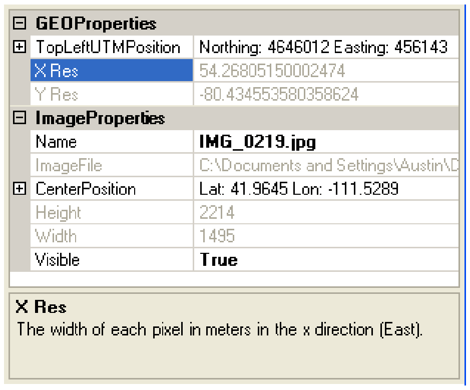
All images have the same group of image properties: image name, where the image is located, the center position of the image, the width and height of the image, and whether or not the image is visible. However, different images will have additional properties. A world file (fig. 3.8(b)), for example, also shows the position of the image's top left corner and the ground resolution of each pixel. The aircraft image has even more properties (fig. 3.8(a)). The GEOProperties of an aircraft image show the position, orientation, and speed of the aircraft when the image was taken. Since these properties are bold, the user can manipulate them to change how the image is orthorectified. These are also the properties where the user could convert images without aircraft data to aircraft images. If the GEOProperties are changed, the difference between the original and current property values are shown in the Errors properties. For example, if the user changes the Latitude property, the LatitudeError property will show the difference. Furthermore, the PositionErrorMagnitude property will show the distance from the original position and the current position, and the PositionErrorDirection property will show the orientation of the current position from the original position.



⊟ **Errors**	
LatitudeError	0°
LongitudeError	0°
PositionErrorMagnitude	0
PositionErrorDirection	360°
AltitudeError	0
RollError	0°
PitchError	0°
YawError	-9°
⊟ **GEOProperties**	
⊞ UTMPosition	Northing: 4645690 Easting: 455834
Latitude	**41.961956°**
Longitude	**-111.5329742°**
Altitude	**136.937080078125**
Roll	**0.2864782276547°**
Pitch	**5.729564553094°**
Yaw	**-119.695187165775°**
Speed	19.03
Heading	-127.7509°
⊟ **ImageProperties**	
Name	**IMG_0227.JPG**
ImageFile	C:\Documents and Settings\Austin\Des
⊞ CenterPosition	Lat: 41.9619 Lon: -111.5331
Height	2448
Width	3264
Visible	**True**

Altitude
The Altitude of the aircraft.

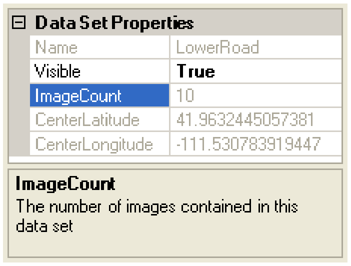
(a) Aircraft Image Properties



⊟ **GEOProperties**	
⊞ TopLeftUTMPosition	Northing: 4646012 Easting: 456143
X Res	54.26805150002474
Y Res	-80.434553580358624
⊟ **ImageProperties**	
Name	**IMG_0219.jpg**
ImageFile	C:\Documents and Settings\Austin\D
⊞ CenterPosition	Lat: 41.9645 Lon: -111.5289
Height	2214
Width	1495
Visible	**True**

X Res
The width of each pixel in meters in the x direction (East).

(b) World File Properties



⊟ **Data Set Properties**	
Name	LowerRoad
Visible	**True**
ImageCount	10
CenterLatitude	41.9632445057381
CenterLongitude	-111.530783919447

ImageCount
The number of images contained in this data set

(c) Dataset Properties

Fig. 3.8: Examples of the Property Window.

3.4 Advanced Dataset Operations

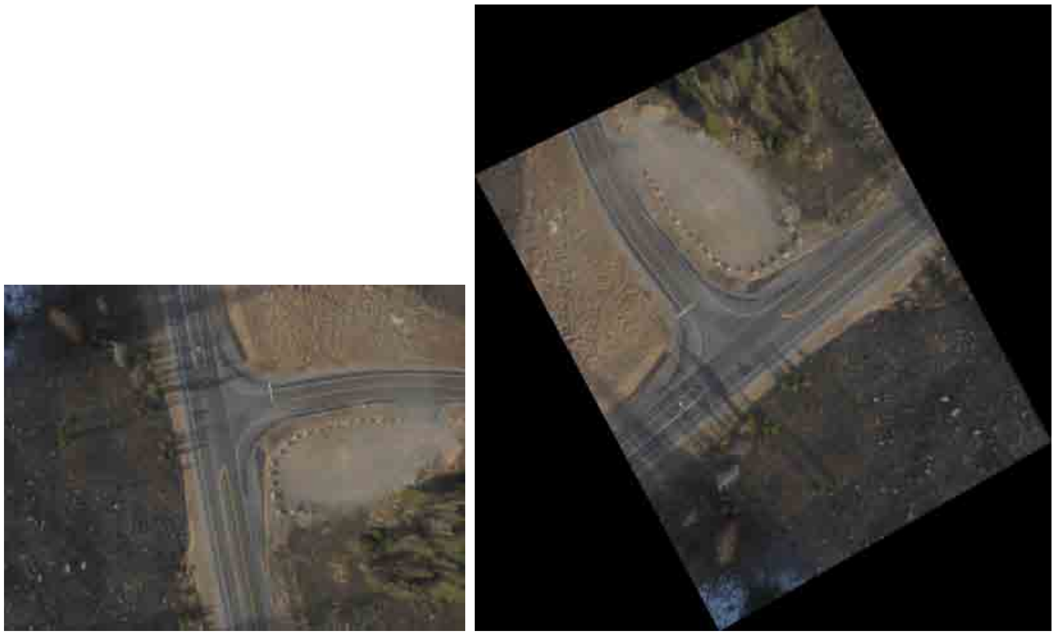
3.4.1 Exporting To World Files

Not only can gRAID import world files, but it can also export them. A dataset can be exported to world files by right clicking on the dataset and selecting Export To WF in the Popup menu. The user will then need to select where to store the world files, the image format and whether or not to crop them. Figure 3.9(a) shows an original image which has been converted into a world file (fig. 3.9(b)). Figure 3.9(c) shows a cropped world file to eliminate the black around the image. The choice of cropping the image or not is given in that cropping the image reduces the coverage and may produce gaps in the map.

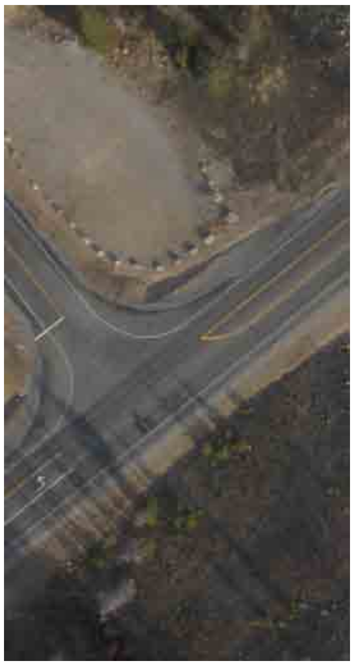
3.4.2 OrthoHelper

Currently, errors in the aircraft sensors prevent the image from being perfectly geo-referenced. OrthoHelper (fig. 3.10) is a tool which helps correct this by helping the user manipulate the aircraft data associated with the aerial photo. While comparing the aerial image with a background image, the user can move, scale, and transform the aerial image until the two images match. The UAV position can be changed using the controls in the position section of OrthoHelper. The North (N), South (S), East (E), and West (W) buttons move the image in their respective directions. The up (U) and down (D) buttons change the altitude of the UAV and scale the size of the image. The orientation of the aircraft can also be changed using the controls in the attitude section of OrthoHelper. The effects of the orientation of the aircraft on the image are shown in fig. 3.11. The roll and pitch will make one side of the image larger than the other. Yaw simply rotates the image.

As the aircraft data is changed, the text boxes in the Accumulated Values section displays the difference between the original aircraft values and the current. The user can use the text boxes to enter in the desired changes instead of using the buttons. Accumulated Values can also be copied and pasted into other images and datasets using the “Copy” and “Paste” buttons. The “Hide” button hides and shows the selected image and the “Restore” button restores the current aircraft values to the original.



(a) Original Image (b) World File



(c) Cropped World File

Fig. 3.9: Example of world files.

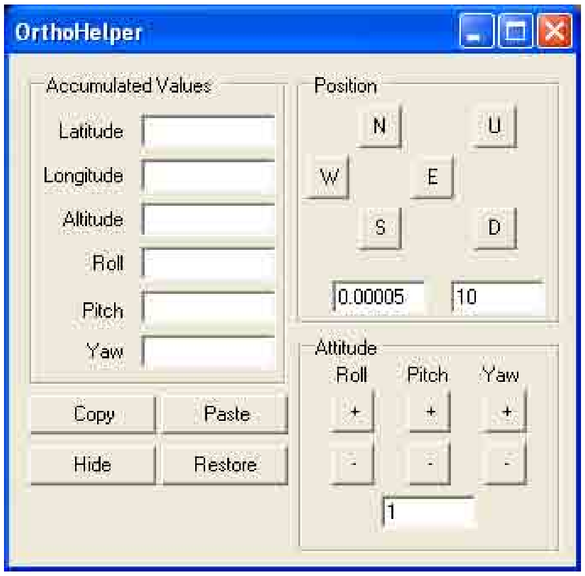
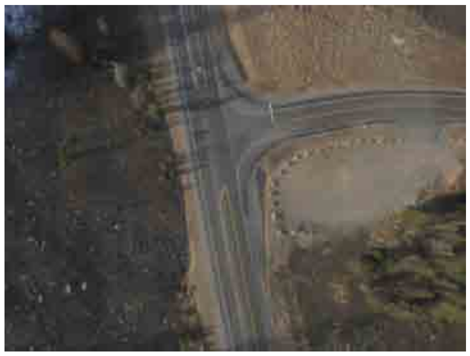
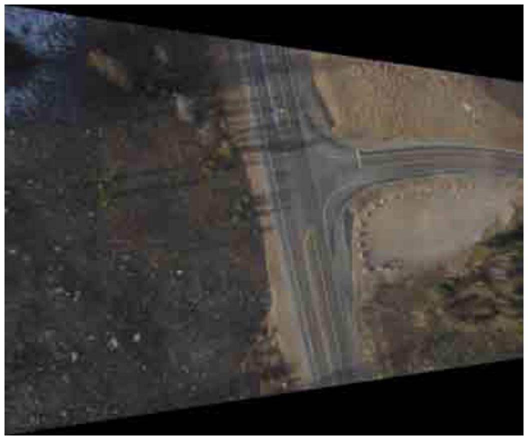


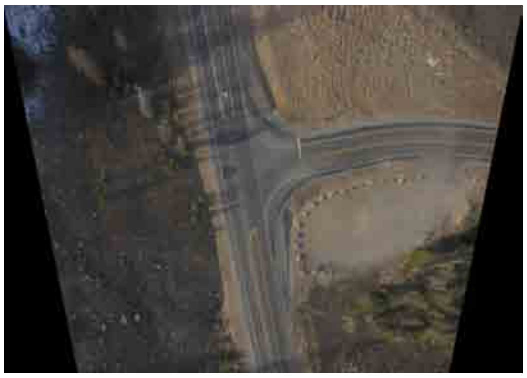
Fig. 3.10: The OrthoHelper tool.

3.5 Real-Time Image Processing and Display

The Video Stream Tool (fig. 3.12) is used to import real-time images into gRAID. It grabs images from a frame grabber, processes, georeferences, and adds the images to a dataset. Currently this is only compatible with a Sensoray 2255 4 channel USB frame grabber. In the Video Stream Control section of the Video Stream Tool, the Start button begins grabbing images from the frame grabber and displaying the video on the Video Display. The channel displayed in the Video Display is selected using the Display Channel combo box. In addition to starting the video stream, after the Start button is pressed, the Image Processing section becomes accessible. Each check box in the Image Processing section enables a different image processing step for each active video stream channel. The active channels are specified by the selected aircraft in the Aircraft List. The Process Images check box corrects the stream of images for radial distortion. The Georeference check box first adds one dataset for each selected aircraft to the top dataset. Additional datasets, one for each active channel, are also added to these new aircraft datasets. Then after all the new datasets are created, the images are georeferenced, added to the datasets, and displayed on the World Window. Georeferencing can be paused without creating a whole new set of datasets by pushing the play/pause button next to the Georeference check box.


(a) Original Image


(b) Roll = 25°


(c) Pitch = 25°

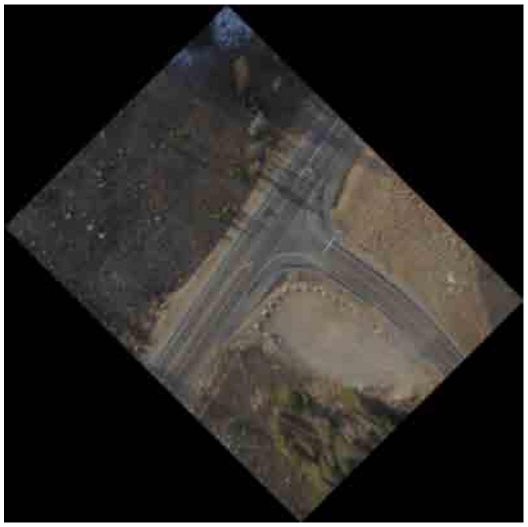

(d) Yaw = 45°

Fig. 3.11: Effects of the orientation of the aircraft.

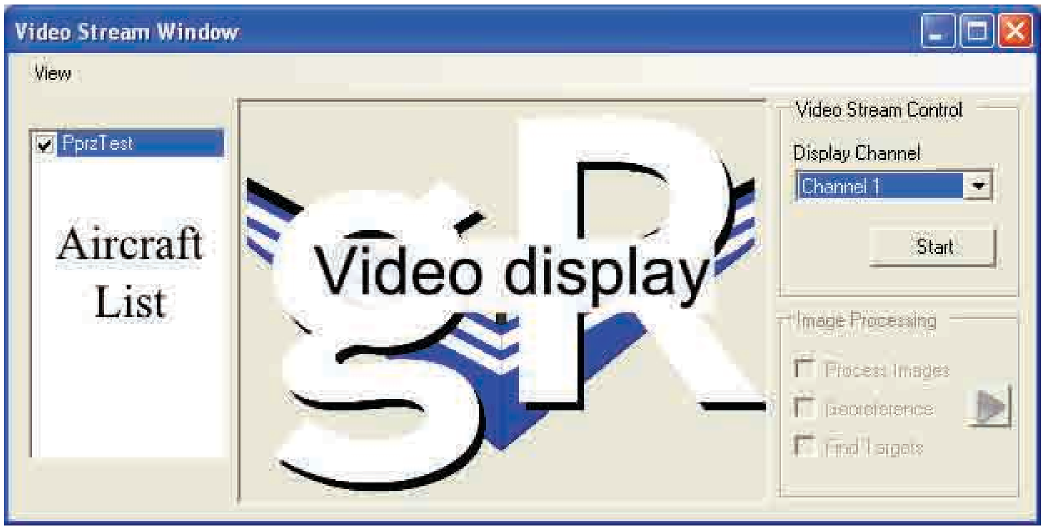
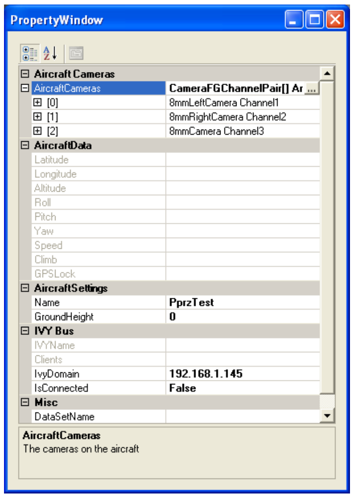


Fig. 3.12: Video Stream Tool.

3.5.1 Aircraft Object

Aircraft objects can be added to gRIAD through the gRAID Object Dialog. Each aircraft object contains important information and tools for real-time image processing and display. The properties of an aircraft object (fig. 3.13) can be viewed by right clicking on the aircraft in the Aircraft List section in the Video Stream Tool or by going to the gRIAD Object Dialog. Basic properties of an aircraft object include the name, ground altitude, and dataset name. The dataset name is the name of the dataset created by the Video Stream Tool and can be changed while the images are being downloaded and displayed. This gives the user the ability to organize the data in real-time instead of after the flight. If the dataset name is not specified, the name of the aircraft will be used instead. The cameras on the aircraft and the frame grabber channel they are paired with are listed under the AircraftCamera properties. The cameras selected here are the same cameras created in the gRIAD Object Dialog and contain the necessary information for georeferencing the images. The channels paired with each of the cameras are the active channels for this aircraft.

To successfully georeference each image sent down from the aircraft, the position and orientation of the aircraft will need to known once the image is grabbed from the frame grabber. This can be done by connecting the computer running World Wind to the Paparazzi


PropertyWindow

Aircraft Cameras	
AircraftCameras	**CameraFGChannelPair[] Ar**
[0]	8mmLeftCamera Channel1
[1]	8mmRightCamera Channel2
[2]	8mmCamera Channel3
AircraftData	
Latitude	
Longitude	
Altitude	
Roll	
Pitch	
Yaw	
Speed	
Climb	
GPSLock	
AircraftSettings	
Name	**PprzTest**
GroundHeight	**0**
IVY Bus	
IVYName	
Clients	
IvyDomain	**192.168.1.145**
IsConnected	**False**
Misc	
DataSetName	

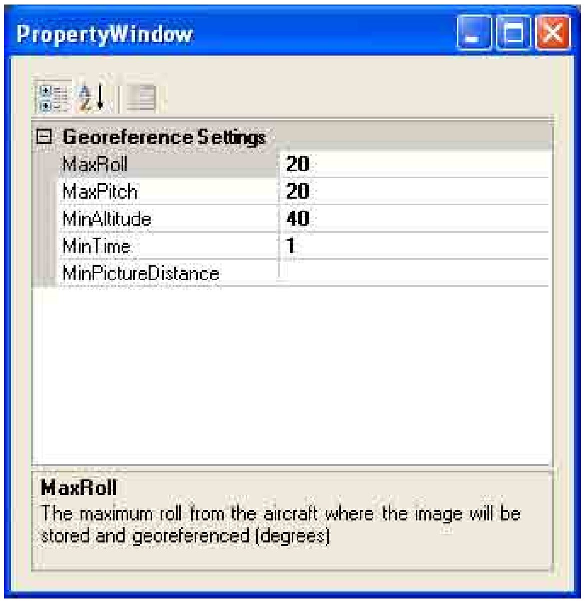
AircraftCameras
The cameras on the aircraft

Fig. 3.13: The Aircraft Property Window.

GCS computer though Ethernet and by listening to the Ivy bus. gRAID is able to listen to the Ivy bus by providing the IvyDomain address and selecting true for the IsConnected property. Once successfully connected, the AircraftData properties will begin updating with the current UAV data from Paparazzi. The aircraft data will update automatically in the Aircraft Property Window if one of the aircraft data properties is selected.

3.5.2 Video Stream Properties

Like the GDL Dialog, the Video Stream Tool also has a way to filter out specific images based on aircraft data. In the Video Stream Properties Dialog (fig. 3.14), the user can specify the maximum roll and pitch, minimum altitude, minimum sample time, and the minimum distance between sampled images.



Georeference Settings

MaxRoll	20
MaxPitch	20
MinAltitude	40
MinTime	1
MinPictureDistance	

MaxRoll
The maximum roll from the aircraft where the image will be stored and georeferenced (degrees)

Fig. 3.14: Video stream properties.

Chapter 4
gRAID Image Processing

Figure 4.1 shows how gRAID processes each image. First some basic processing is performed on the image. The radial distortion is corrected and the image channels can be separated and converted into grey scale images. After the basic processing, the image is georeferenced and the position of each corner in the image is found. The corner positions can then be used to either draw the image on World Wind or to export the image to a world file.

All the basic image processing, including calibration, is done with an an open source computer vision programming library called OpenCV [27].

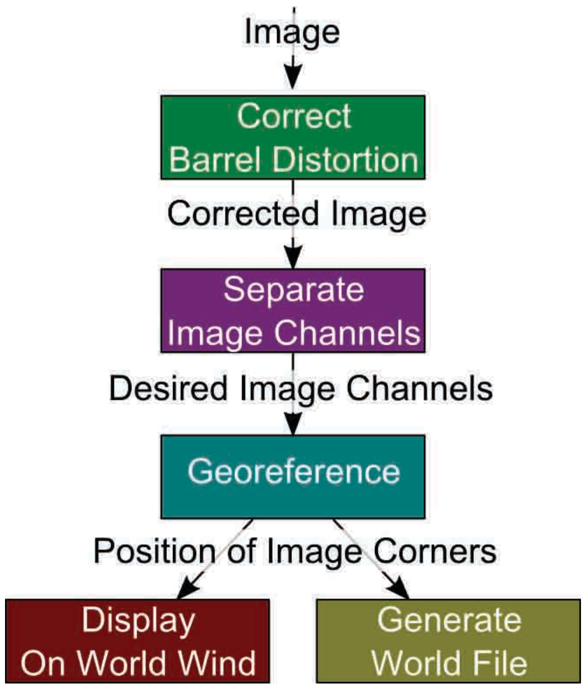


Fig. 4.1: Flow diagram of the image processing.

4.1 Camera Calibration

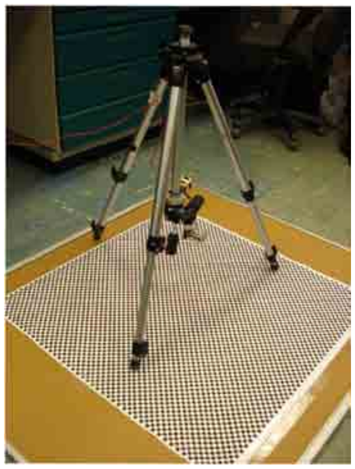
To georeference the images, it is important to find the FOV of the camera because it is used to calculate the initial corner positions of the image. In many cases, the FOV is provided by the camera manufacture or can be calculated using sensor size and focal length. However this may not be accurate enough, especially if the image is being corrected for radial distortion. Initially, the simple solution displayed in fig. 4.2(a) was used to find the FOV. The camera was mounted on a tripod looking down at a 90x90cm grid (fig. 4.2(b)) composed of 8,100 1cm black and white squares. The FOV is found by measuring the distance between the camera and the grid, taking a picture, and counting the number if squares across the image horizontally and vertically. When correcting the image for radial distortion, the image should be corrected before counting the squares.

The radial distortion of the camera can be found using a set of camera calibration OpenCV functions and a small black and white grid. By taking pictures of the grid at different orientations to the camera (fig. 4.3), OpenCV can detect the corners of the grid in each image, find their positions, and find the intrinsic properties and the distortion coefficients. These intrinsic properties include the location of the principle point (c_x, c_y) and the scale factors in the image axes (f_x, f_y). The distortion coefficients not only include the radial distortion coefficients (k_1 and k_2), but also the translational distortion coefficients (p_1 and p_2).

4.2 Radial Distortion

Radial Distortion is caused by the camera lens. As shown by fig. 4.4, radial distortion decreases the magnification of the image as the radius from the optical axis increases [28]. Wide angle (fish-eye) lenses, with short focal lengths, induce the most radial distortion and is easily noticed. Even though lenses with longer focal lengths may not create enough radial distortion to notice, the calibration method will still detect it and is still important to correct.

The radial distortion is easily corrected using OpenCV and the Undistort2 function. The Undistort function is designed to remove all lens distortion including translational


(a) Apparatus for Finding FOV

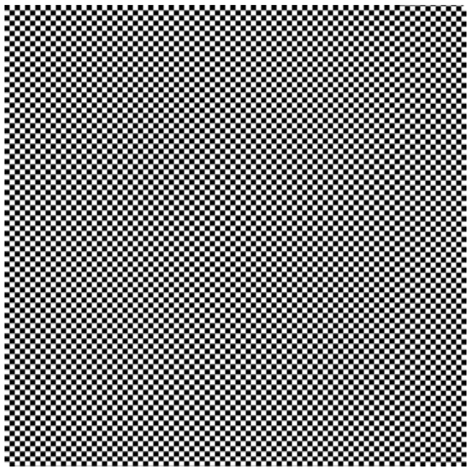

(b) 90x90cm Grid

Fig. 4.2: Calibrating the FOV.

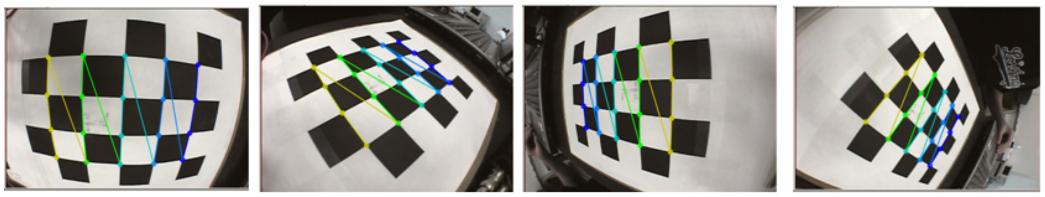


Fig. 4.3: Grid samples for camera calibration.

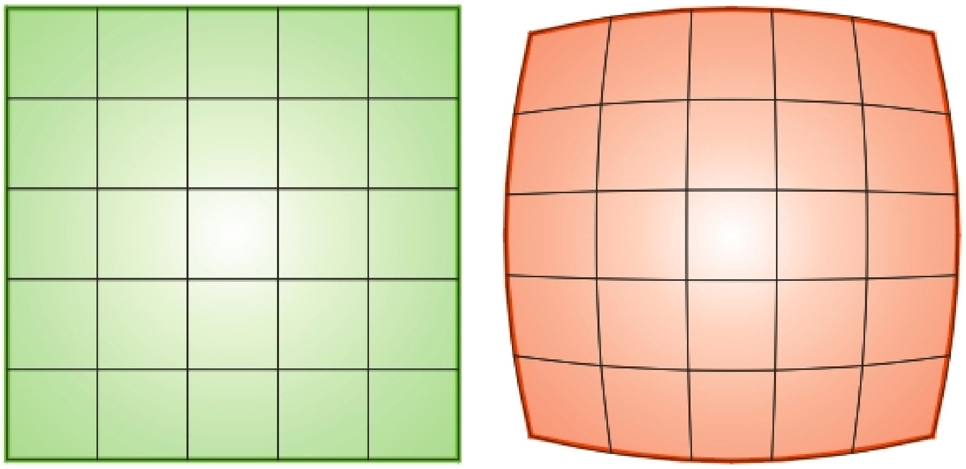

(a) Before Radial Distortion (b) After Radial Distortion

Fig. 4.4: Radial distortion example.

distortion. Given the distorted image, the camera intrinsic values, and the distortion coefficients from the calibration, Undistort2 will output a new corrected image.

4.3 Image Channel Separation

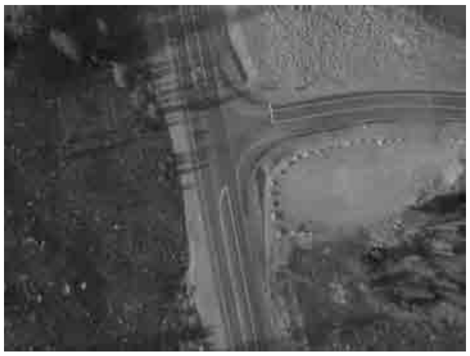
In some cases, the user may only be interested in a single channel from the image (red, green, or blue). gRIAD has the capabilities to separate each of these channels and convert them into a grey scale image (fig. 4.5). This is useful when the user is only interested in the reflectance in a specific band of the spectrum. For example, with the NIR camera, the red pixel is the only one which responds to the NIR light. Therefore, it is useful to separate the red channel and convert it into grey scale.

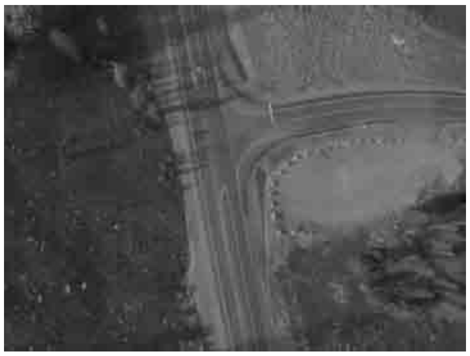
4.4 Image Georeferencing

4.4.1 Coordinate Systems

To georeference the aerial images, a few coordinate systems must first be defined. Figure 4.6 shows the two coordinate systems on the aircraft: the body frame and the camera frame. The origin of the body frame is located at the center of gravity. The x axis points through the nose, the y axis points out the right wing, and the z axis points down.

(a) Original Image


(b) Red Image


(c) Green Image

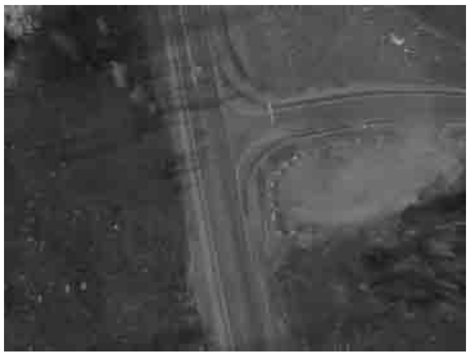

(d) Blue Image

Fig. 4.5: Color conversion example.

The origin of the camera frame is at the focal point of the camera. However, since the distance from the focal point to the center of gravity is smaller than the orthorectification accuracy, we will assume that the focal length and the center of gravity are located at the same position. The axes of the camera frame are rotated by ϕ_c, θ_c, and ψ_c with respect to the body frame.

The inertial frame for the UAV is the North, East, Down (NED) coordinate system (fig. 4.6). The x axis points towards the north, the y axis points towards the east and the z axis points down. The orientation of the UAV with respect to the NED frame is given by ϕ, θ, and ψ. For the purposes of georeferencing the images (not for navigation), we are going to assume the origin of the NED is also at the aircraft center of gravity.

To georeference the images with respect to geodetic coordinates, another important coordinate system is the earth-centered earth-fixed (ECEF) coordinate system (fig. 4.7). In the ECEF coordinate system, the z axis passes through the north pole, the x axis passes through the equator at the prime meridian, and the y axis passes through the equator at 90° longitude. The ECEF system is the same system used by GPS except GPS is given in spherical coordinates not Cartesian.

4.4.2 Generating the Image Corners

Any point in an image can be rotated from the camera frame to the ECEF coordinate system in order to find where it is located on the earth. However, it is only necessary to find the location of the four corners of the image in order to georeference it. Assuming the origin is at the focal point and the image is on the image plane, eq. (4.1) can be used to find the four corners of the image. As defined in fig. 4.8, FOV_x is the FOV around the x axis, FOV_y is the FOV around the y axis, and f is the focal length.

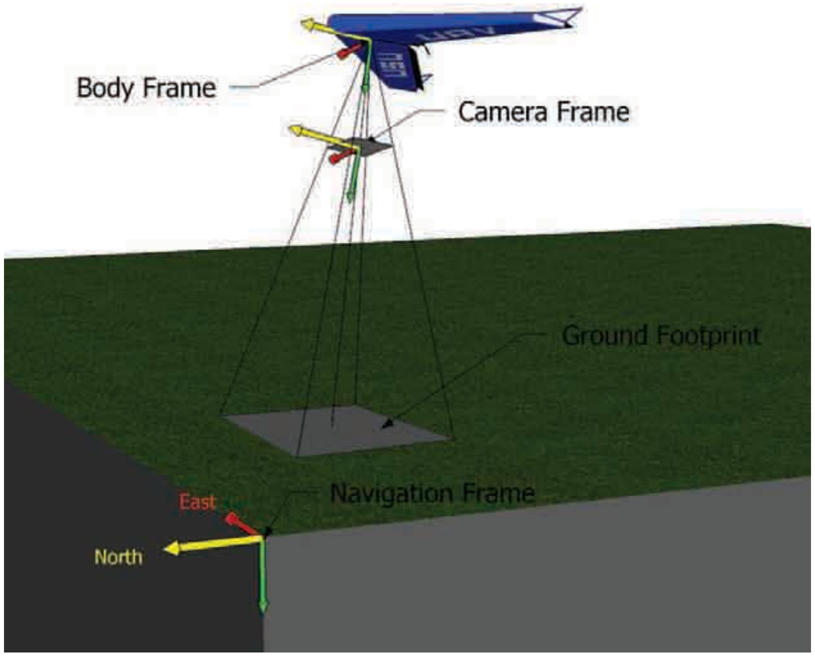


Fig. 4.6: Aircraft coordinate systems.

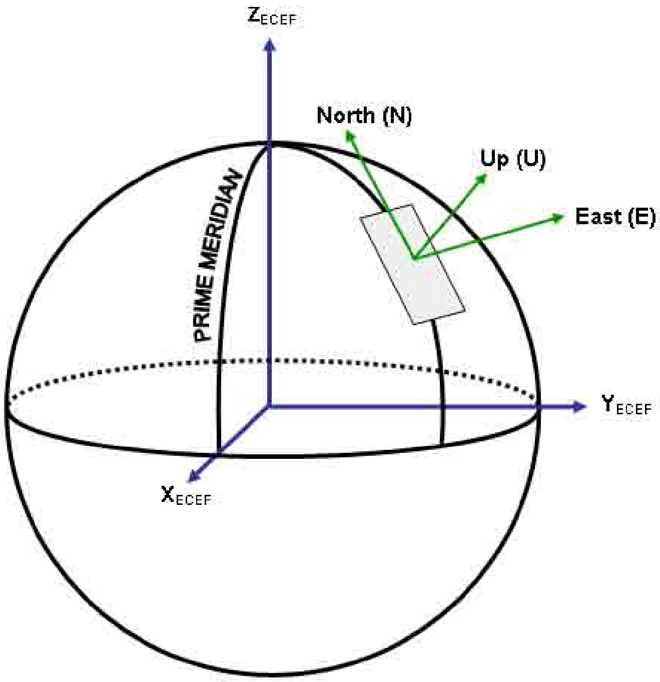


Fig. 4.7: Earth coordinate systems.

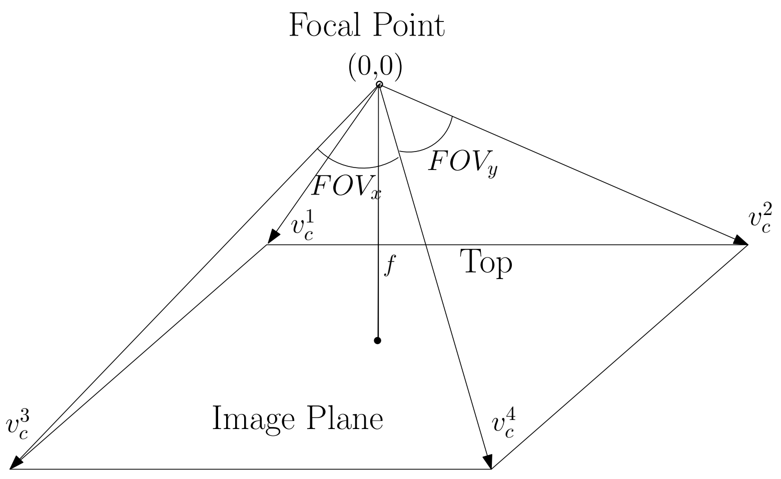


Fig. 4.8: Definition of initial image corners.

$$v_c^1 = \begin{bmatrix} f\tan(FOV_y/2) & -f\tan(FOVx/2) & f \end{bmatrix} \tag{4.1}$$

$$v_c^2 = \begin{bmatrix} f\tan(FOV_y/2) & f\tan(FOVx/2) & f \end{bmatrix} \tag{4.2}$$

$$v_c^3 = \begin{bmatrix} -f\tan(FOV_y/2) & -f\tan(FOVx/2) & f \end{bmatrix} \tag{4.3}$$

$$v_c^4 = \begin{bmatrix} -f\tan(FOV_y/2) & f\tan(FOVx/2) & f \end{bmatrix} \tag{4.4}$$

4.4.3 Rotating into Navigation Frame

To rotate the corners into the navigation frame, they first need to be rotated into the body frame. The Euler angles with respect to the body frame are given by ϕ_c, θ_c, and ψ_c, and can be used to create a rotation matrix R_c^b which rotates a vector in the body frame to the camera frame.

$$R_c^b = R_{xyz}(\phi_c, \theta_c, \psi_c) \tag{4.5}$$

To rotate from the camera frame to the body frame, the transpose of R_c^b is used.

$$R_b^c = (R_c^b)^T = R_{zyx}(-\psi_c, -\theta_c, -\phi_c) \tag{4.6}$$

The same rotation matrix is used, with ϕ, θ, and ψ, to rotate from the body into the navigation frame.

$$R_n^b = (R_b^n)^T = R_{zyx}(-\psi, -\theta, -\phi) \tag{4.7}$$

Now each corner is rotated from the camera frame into the navigation frame using eq. (4.8).

$$v_n^i = R_n^b R_b^c v_c^i \tag{4.8}$$

4.4.4 Transforming into the ECEF Coordinate System

Now that the corners are in the NED coordinate system, they are scaled to the ground to find their appropriate magnitude (assuming flat earth) where h is the height of the UAV above ground and $v_n^i(z)$ is the z component of v_n^i.

$$v_n^i = v_n^i \frac{h}{v_n^i(z)} \tag{4.9}$$

The next step is to rotate the image corners into the ECEF coordinate system. This is done with another rotation matrix and the latitude (λ) and longitude (α) of the UAV.

$$R_w^n = R_{zyy}(-\alpha, \frac{\pi}{2}, \lambda) \tag{4.10}$$

$$v_w^i = R_w^n v_n^i \tag{4.11}$$

After the corners are rotated into the ECEF coordinate system, they are located in the center of the earth and need to be translated up to the position of the UAV in cartesian coordinates (p).

$$v_w^i = v_w^i + p \tag{4.12}$$

Now v_w^i represents the position of each of the image corners, in cartesian coordinates, projected on the earth.

4.5 Drawing the Images

The image can now be drawn on World Wind by creating a 3D surface and overlaying the image on it. A mesh of points, like in fig. 4.9, is found using the corners of the images and represents a surface by defining multiple, conjoining triangles inside it.

The mesh for the surface of the image is defined by the corners of the image and the mesh number (n), which is the desired number of points along each edge of the image. Each mesh point (m_j) along the edges of the image can be defined by a vector (v_j) incrementally rotated between two adjoining corners (v_i and v_f), where j goes from 1 to $n-1$ (fig. 4.10).

Equation (4.13) is used to find v_j by rotating v_i by a rotation matrix, which is a function of the vector (v_x) and the angle (θ_j). v_x is a perpendicular vector to v_i and v_f, and θ_j is incremented by $\frac{\theta}{(n-1)}$ for every iteration until $\theta_j = \theta$. θ is the angle between v_i and v_f.

$$v_j = R(v_x, \theta_j)v_i \tag{4.13}$$

$$v_x = v_i \times v_f \tag{4.14}$$

$$\theta_j = \frac{\theta}{(n-1)}(j+1) \tag{4.15}$$

To find the points inside the image surface, the mesh points along the edges of the surface can be used as v_i and v_f and eq. (4.13) can be recalculated (fig. 4.11).

4.6 Generating World Files

One nice thing about displaying aerial images on World Wind is that its 3D environment automatically corrects the image for perspective projection. That is, the image is automatically transformed from the perspective of the camera, which has the illusion of depth, to an image which is evenly spatially distributed on the ground [29]. Therefore when

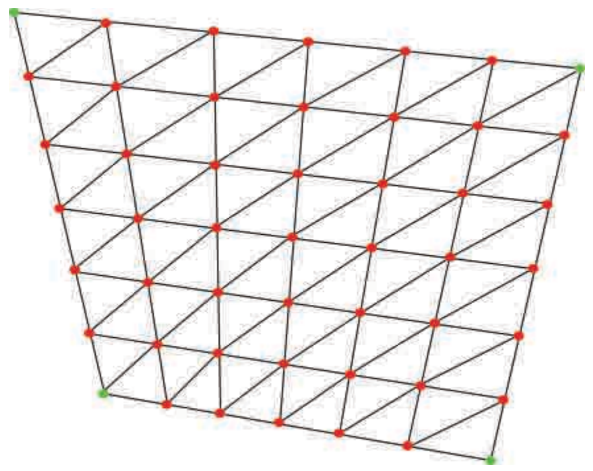


Fig. 4.9: Example of image surface created by triangles ($n = 7$).

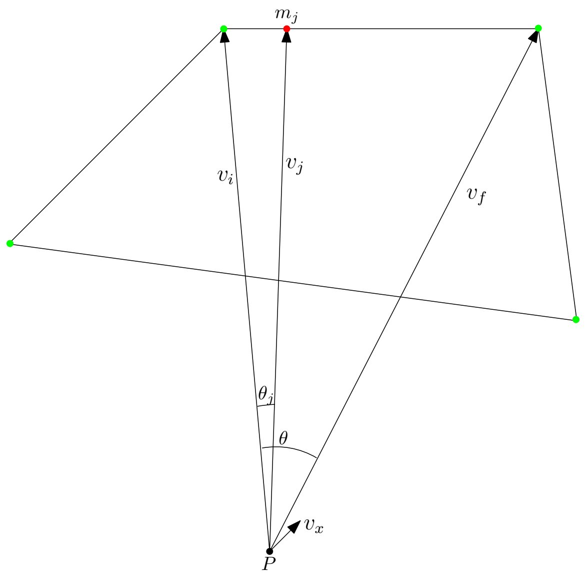


Fig. 4.10: Calculating the mesh points.

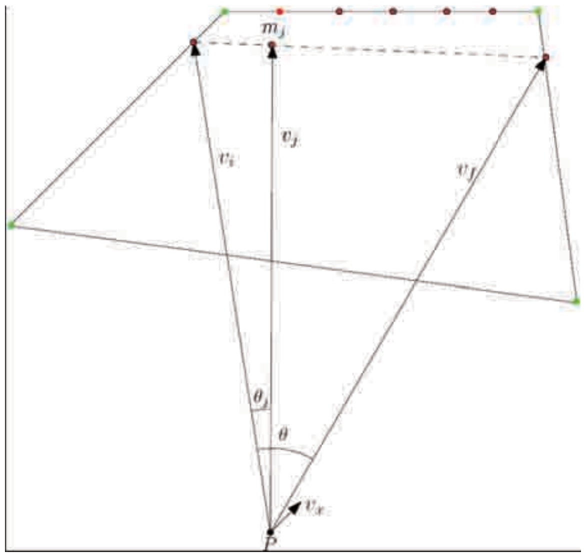


Fig. 4.11: Calculating the mesh points inside the image.

generating a world file, which must be evenly distributed, the raw image must be perspectively transformed. This is done by mapping the position of each pixel from the raw image to the new corrected image with a matrix. This matrix can be found by providing the corner positions of the original image and the desired corner positions of the transformed image, and finding out how the original corners are mapped to the transformed corners. This can be done using an OpenCV function called WarpPerspectiveQMatrix. Once the matrix is found, it can be applied to the image using WarpPerspective. Figure 4.12 shows an example of a transformed image.

To find the perspective transformation matrix, the corners found with eq (4.8) can be used. Instead of projecting them down to the ground they are projected onto the focal plane.

$$v_f^i = v_n^i \frac{f}{v_n^i(z)} \tag{4.16}$$

v_f^i represents the corner positions after the perspective transformation and $v_i^i = v_c^i$ represents the corner positions before. However, they need to be rotated and translated into the image coordinate system and converted into pixel values. In the image coordinate system, the origin is at the top left corner, the x axis runs across the top of the image, and the y axis runs down along the side (fig. 4.13).

v_i^i and v_f^i are transformed into the image coordinate system by rotation around the z axis $\frac{\pi}{2}$ and translation by their maximum x and y values.

$$v_i^i = R_z(\frac{\pi}{2})v_i^i - v_i^{max} \tag{4.17}$$

$$v_f^i = R_z(\frac{\pi}{2})v_f^i - v_f^{max} \tag{4.18}$$

The corners can then be converted into pixel values by finding how many pixels per meter are in the image (P_x and P_y) and multiplying by it. w and h are the pixel width and height of the original image.

$$P_x = wv_i^3(x) \tag{4.19}$$

$$P_y = hv_i^3(y) \tag{4.20}$$

$$v_i^i(x) = v_i^i(x)P_x \tag{4.21}$$

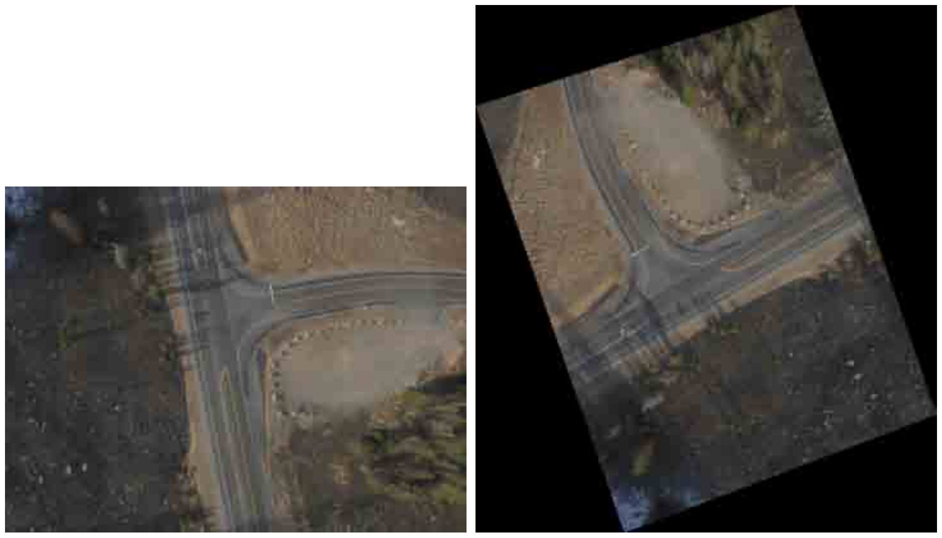
$$v_i^i(y) = v_i^i(y)P_y \tag{4.22}$$

$$v_f^i(x) = v_f^i(x)P_x \tag{4.23}$$

$$v_f^i(y) = v_f^i(y)P_y \tag{4.24}$$

The corners in v_i^i and v_f^i can now be used in WarpPerspectiveQMatrix (neglect the z value) to find the transformation matrix, which can be used in WarpPerspective to transform the image.

To include the position information with this image, a text file is included in the same directory which shares the same name as the image, but has a different extension. Here is how the text file is organized.



(a) Original Image (b) Transformed Image

Fig. 4.12: Perspective transformation example.

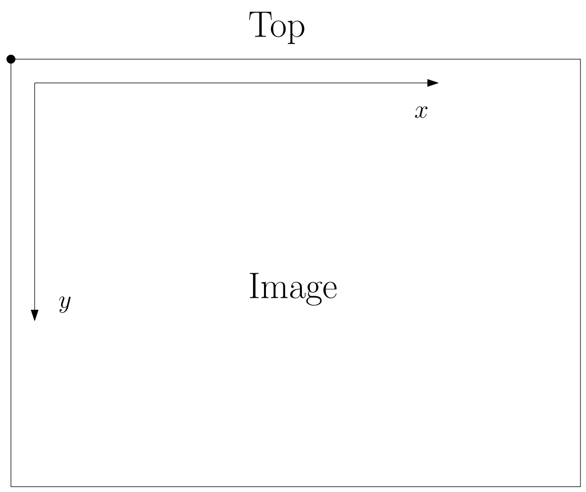


Fig. 4.13: Image coordinate system.

1. Line 1: Pixel size in the Easting direction (meters/pixel)
2. Line 2: Rotation about y-axis (0 when using UTM)
3. Line 3: Rotation about x-axis (0 when using UTM)
4. Line 4: Pixel size in the Northing direction (meters/pixel)
5. Line 5: UTM Easting location of top left corner
6. Line 6: UTM Northing location of top left corner

To find the location of the top left corner of the image, convert all the corners (v_w^i) to UTM and find the maximum Northing and the minimum Easting value. This point is the top left corner. For the pixel sizes, find the difference between the minimum and maximum Northing and Easting values. Then divide these differences by the number of pixels, in the transformed image, in each respective direction. These values are the pixel sizes.

Chapter 5
Conclusion and Suggestions for Future Research

5.1 Summary of Contributions

A low-cost, small, autonomous remote sensing platform has been developed to provide remote sensing data to more people. Images systems have also been developed for the platform to provide the images and to synchronize the images with the data from the aircraft. With the images and the aircraft data, a program has also been developed to process, georeference and display the images on a 3D world viewer called World Wind. The user friendly program (gRAID) not only imports images taken from the UAV, but also world files and images without any aircraft data. Tools have also been provided with gRAID which allow the user to manipulate the aircraft data and to retrieve the images from the aircraft in real-time. Camera calibration data can also be entered into gRAID to increase the orthorecification accuracy by correcting for distortion like radial distortion. To make mosaics and process the images further using conventional GIS software, the images can also be exported to world files.

5.2 Suggestions

Suggestions for further work includes using better calibration techniques to characterize the cameras better (including a spectral response), and improving orthorectification by calibrating aircraft sensors and finding ways to get better GPS quality. In addition, a low-cost thermal image system should be developed to broaden the range of applications for AggieAir.

References

[1] T. T. H. Pham and D. C. He, “How do people perceive the city’s green space? a view from satellite imagery (in Hanoi, Vietnam),” in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium IGARSS 2008*, vol. 3, pp. III–1228–III–1231, 7–11 July 2008.

[2] M. Yan, L. Ren, X. He, and W. Sang, “Evaluation of urban environmental quality with high resolution satellite images,” in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium IGARSS 2008*, vol. 3, pp. III–1280–III–1283, 7–11 July 2008.

[3] J. Everitt, C. Yang, and C. Deloach, “Remote sensing of Giant Reed with Quickbird satellite imagery,” *Journal of Aquatic Plant Management*, vol. 43, pp. 81–85, 2006.

[4] J. A. Voogt and T. R. Oke, “Thermal remote sensing of urban climates,” *Remote Sensing of Environment*, vol. 86, no. 3, pp. 370 – 384, 2003.

[5] A. Murakami and A. Hoyano, “Study on urban heat island phenomenon in a local small city of Japan using airborne themal image,” in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium IGARSS 2008*, vol. 3, pp. III–1354–III–1357, 7–11 July 2008.

[6] W. H. Kraft, “Improved transportation management and operations through the use of remote sensing,” in *Proceedings of the Symposium on Integrating Remote Sensing at the Global, Regional, and Local Scale*, 2002.

[7] G. Palubinskas, F. Kurz, and P. Reinartz, “Detection of traffic congestion in optical remote sensing imagery,” in *Proceedings of the IEEE International Geoscience and Remote Sensing Symposium IGARSS 2008*, vol. 2, pp. II–426–II–429, 7–11 July 2008.

[8] S. P. Hoogendoorn, H. J. Van Zuylen, M. Schreuder, B. Gorte, and G. Vosselman, “Microscopic traffic data collection by remote sensing,” *Journal of the Transportation Research Board*, vol. 1855, pp. 121–128, 2003.

[9] E. Underwood, S. Ustin, and D. DiPietro, “Mapping nonnative plants using hyperspectral imagery,” *Remote Sensing of Environment*, vol. 86, no. 2, pp. 150 – 161, 2003.

[10] D. K. Fisher, J. Hinton, M. H. Masters, C. Aasheim, E. S. Butler, and H. Reichgelt, “Improving irrigation efficiency through remote sensing technology and precision agriculture in SE georgia,” in *2004 ASAE Annual Meeting*, 2004.

[11] S. G. Bajwa and E. D. Vories, “Spectral response of cotton canopy to water stress,” in *Proceedings of the ASAE Annual Meeting*, July 2006.

[12] D. J. Hunsaker, P. J. Pinter Jr., E. M. Barnes, J. C. Silvertooth, and J. Hagler, “Scheduling cotton irrigations using remotely-sensed basal crop coefficients and FAO-56,” in *Proceedings of the ASAE Annual Meeting*, Aug. 2004.

[13] P. J. Pinter Jr., J. L. Hatfield, J. S. Schepers, E. M. Barnes, M. S. Moran, C. S. T. Daughtry, and D. R. Upchurch, “Remote sensing for crop management,” *Photogrammetric Engineering and Remote Sensing*, vol. 69, no. 6, pp. 647–664, June 2003.

[14] J. A. J. Berni, P. J. Zarco-Tejada, L. Suarez, and E. Fereres, “Thermal and narrowband multispectral remote sensing for vegetation monitoring from an unmanned aerial vehicle,” *IEEE Transactions on Geoscience and Remote Sensing*, vol. 47, no. 3, pp. 722–738, Mar. 2009.

[15] H. Eisenbeiss, K. Lambers, and M. Sauerbier, “Photogrammetric recording of the archaeological site of Pinchango Alto (Palpa, Peru) from a mini helicopter (UAV),” in *Proceedings of the 33rd CAA Conference*, 2005.

[16] L. F. Johnson, S. R. Herwitz, B. M. Lobitz, and S. E. Dunagan, “Feasibility of monitoring coffee field ripeness with airborne multispectral imagery,” *Applied Engineering in Agriculture*, vol. 20, pp. 845–849, 2004.

[17] L. F. Johnson, S. R. Herwitz, S. E. Dunagan, B. M. Lobitz, D. V. Sullivan, and R. E. Slye, “Collection of ultra high spatial and spectral resolution image data over california vineyards with a small UAV,” in *Proceedings of the 30th International Symposium on Remote Sensing of Environment*, vol. 20, pp. 845–849, 2003.

[18] R. Graml and G. Wigley, “Bushfire hotspot detection through uninhabited aerial vehicles and reconfigurable computing,” in *Proceedings of the IEEE Aerospace Conference*, pp. 1–13, 2008.

[19] D. W. Casbeer, D. B. Kingston, R. W. Beard, and T. W. McLain, “Cooperative forest fire surveillance using a team of small unmanned air vehicles,” *International Journal of Systems Science*, vol. 37, no. 6, pp. 350–360, May 2006.

[20] D. W. Casbeer, R. W. Beard, T. W. McLain, S.-M. Li, and R. K. Mehra, “Forest fire monitoring with multiple small uavs,” in *Proceedings of the American Control Conference 2005*, pp. 3530–3535, 8–10 June 2005.

[21] S. Drake, K. Brown, J. Fazackerley, and A. Finn, “Autonomous control of multiple UAVs for the passive location of radars,” in *Proceedings of the 2005 International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, pp. 403–409, Dec. 2005.

[22] M. Pachter, N. Ceccarelli, and P. Chandler, “Vision-based target geo-location using camera equipped mavs,” in *Proceedings of the 46th IEEE Conference on Decision and Control*, pp. 2333–2338, 2007.

[23] H. Chao, M. Baumann, A. Jensen, Y. Chen, Y. Cao, W. Ren, and M. McKee, “Band-reconfigurable multi-UAV-based cooperative remote sensing for real-time water management and distributed irrigation control,” in *Proceedings of the IFAC World Congress, Seoul, Korea*, July 2008.

[24] Paparazzi UAV, [http://paparazzi.enac.fr/wiki/Main_Page], Jan. 2009.

[25] M. Baumann, "Imager development and image processing for small UAV-based real-time multispectral remote sensing," Master's thesis, Hochschule Ravensburg-Weingarten University of Applied Sciences and Utah State University, Oct. 2007.

[26] World Wind, [http://www.worldwindcentral.com/wiki/Main_Page], Jan. 2009.

[27] OpenCV, [http://opencv.willowgarage.com/wiki/], Jan. 2009.

[28] D. Kaplanek, "Development of a real-time vision-based target recognition and geo-location system using camera equipped autonomous UAV's," Master's thesis, Hochschule Ravensburg-Weingarten University of Applied Sciences and Utah State University, Apr. 2008.

[29] D. Salomon, *Transformations and Projections in Computer Graphics*, ch. Perspective Projection, pp. 71–144. London: Springer, 2006.

Appendices

Appendix A

World Wind Plug-in Development

A.1 Installation

Good instructions on installing World Wind for development can be found here.
http://worldwindcentral.com/wiki/Compiling_the_sources#Compile_the_Code

A.2 The World Wind Plug-in Class

The World Wind plug-in class is the interface which gives the user control over what World Wind displays. To use the plug-in class you only need to create another class which inherits it.

```
using WorldWind;
using WorldWind.Renderable;
using WorldWind.Net;

namespace WorldWind.gRAIDPlugin
{
    public class gRAIDPlugin : WorldWind.PluginEngine.Plugin
    {

    }
}
```

The inherited class then needs to override a few virtual functions of the plug-in class: Load and Unload

```
public override void Load()
{
    Control ToolButtonControl = new Control();
    ToolButtonControl.Visible = false;
    gRAIDButton = new WorldWind.WindowsControlMenuButton("gRAID", "MainIcon.PNG",
        ToolButtonControl);
    ParentApplication.WorldWindow.MenuBar.AddToolsMenuButton(gRAIDButton);

    ToolButtonControl.VisibleChanged += new
        EventHandler(ToolButtonControl_VisibleChanged);

    gRAIDButton.SetPushed(false);
}

public override void Unload()
{
    gRAIDWindow.Close();
}

public static gRAIDForm gRAIDWindow;
Control ToolButtonControl = new Control();
WorldWind.Menu.MenuButton gRAIDButton;
```

These functions are called by World Wind whenever the plug-in is loaded or unloaded. The plug-in can be loaded and unloaded by the user in the Plug-in Load/Unload dialog under the Plug-Ins menu. The plug-in can also be setup to load when World Wind starts up in the Plug-in Load/Unload dialog. The plug-in is also unloaded when World Wind is shut down.

A common thing to do in the Load function is to add a button in the World Wind menubar (the big buttons at the top of the window) and to setup an event handler. Whenever the button is pushed, the event handler will be called. The Unload function is commonly used to clean up the variables declared in your code.

The code below shows an example of the event handler called by the plug-in button when it is pushed. This is where code specific to your application will go. In the case below, a new form is created and displayed on the screen. This form may contain a completely separate user interface which allows the user to perform many different functions. The Tool Button event handler is also a good place to create some key and mouse event handlers for World Wind.

```
private void ToolButtonControl_VisibleChanged(object sender, EventArgs e)
{
    if (gRAIDButton.IsPushed())
    {
        gRAIDWindow = new gRAIDForm(ParentApplication, ProgramDirectory);

        ParentApplication.WorldWindow.KeyDown += new
            KeyEventHandler(gRAIDWindow.WorldWindow_KeyDown);

        gRAIDWindow.FormClosed += new FormClosedEventHandler(gRAIDWindow_FormClosed);

        gRAIDWindow.FitFormToWorldWind();
        gRAIDWindow.Show(ParentApplication);
    }
    else
    {
        gRAIDWindow.Close();
    }
}

void gRAIDWindow_FormClosed(object sender, FormClosedEventArgs e)
{
    gRAIDButton.SetPushed(false);
}

public static gRAIDForm gRAIDWindow;
```

The plug-in can then be used with World Wind in three different ways. The easiest way is to place the .cs file, with a class which inherits the Plug-in class, in the Plug-ins World Wind directory. World Wind contains a compiler which will read and compile the code for you. However if you have multiple files, World Wind will display all of them in the Load/Unload dialog and it might not work. So the second way to use the plug-in with World Wind is to compile the code into a dll and put the dll in the World Wind Plug-ins directory. The third way to use the plug-in with World Wind is to add your new plug-in class to the WorldWind class under the World Wind project in Visual Studio.

A.3 Renderable Objects and Lists

To draw something on World Wind, all you need to do is create a RenderableObject and add it to a RenderableObjectList. A RenderableObjectList is a class within World Wind which contains a list of RenderableObjects. In addition it inherits the RenderableObject class (abstract), and defines some of its virtual functions used by World Wind: Initialize, Update, Render, Dispose. Initialize is called when the RenderableObject is first created, Render is called when World Wind is ready to draw the object, Update is called before Render to update any needed variables and Dispose is called when the object is being deleted. Even though a RenderableObjectList can be used directly to draw on World Wind, they are usually only used to pass instructions to each RenderableObject added to the list. For example, if the Render function of a RenderableObjectList is called, it will in turn call the Render function of each RenderableObject added to the list. Figure A.1 shows the relationship between World Wind, RenderableObjectLists, and RenderableObjects.

In this configuration, A RenderableObjectList can also be used to filter and sort the list of RenderableObjects. For example, the RenderableObjectList can be written to only draw objects which have a true value for the IsOn property. It can also be written to draw the RenderableObject, over which the mouse is hovering, last to make sure it is drawn over top of everything else. Below shows some code which will draw the objects in the list if their IsOn property is true.

```
namespace WorldWind.Renderable
{
   public class RenderableObjectList : RenderableObject
   {
      public override void Initialize(DrawArgs drawArgs)
      {
         if(!this.IsOn)
            return;

         foreach(RenderableObject ro in this.m_children)
         {
            if(ro.IsOn)
               ro.Initialize(drawArgs);
```

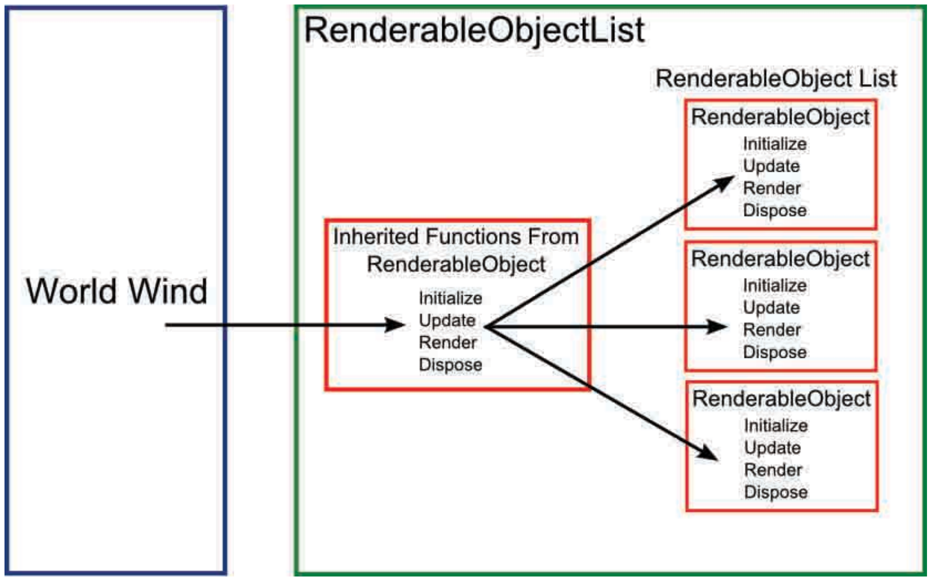


Fig. A.1: RenderableObjectList.

```
        }

        this.isInitialized = true;
    }



    public override void Update(DrawArgs drawArgs)
    {
        if(!this.IsOn)
            return;

        if(!this.isInitialized)
            this.Initialize(drawArgs);



        foreach(RenderableObject ro in this.m_children)
        {
            if(ro.ParentList == null)
                ro.ParentList = this;

            if(ro.IsOn)
            {
                ro.Update(drawArgs);
            }
        }
    }

    public override void Render(DrawArgs drawArgs)
    {
        if(!this.IsOn)
            return;

        foreach(RenderableObject ro in this.m_children)
        {
            if(ro.IsOn)
                ro.Render(drawArgs);
        }
    }

    public override void Dispose()
    {
        this.isInitialized = false;

        foreach(RenderableObject ro in this.m_children)
            ro.Dispose();

        if(m_RefreshTimer != null && m_RefreshTimer.Enabled)
            m_RefreshTimer.Stop();
    }
  }
}
```

After the RenderableObjectList calls the Render function of a RenderableObject and passes the drawArgs, the RenderableObject can use the drawArgs to draw on World Wind. The code below shows the render function of a RenderableObject that draws an image on the surface of the earth.

```
public override void Render(DrawArgs drawArgs)
{
    if (isInitialized && Visible)
    {
        drawArgs.device.SetTexture(0, _ImageTexture);
        drawArgs.device.RenderState.ZBufferEnable = true;
        drawArgs.device.Clear(ClearFlags.ZBuffer, 0, 1.0f, 0);
        drawArgs.device.Transform.World = Matrix.Translation(
                (float)-drawArgs.WorldCamera.ReferenceCenter.X,
                (float)-drawArgs.WorldCamera.ReferenceCenter.Y,
                (float)-drawArgs.WorldCamera.ReferenceCenter.Z
                );

        drawArgs.device.TextureState[0].ColorOperation = TextureOperation.SelectArg1;
        drawArgs.device.TextureState[0].ColorArgument1 =
            TextureArgument.TextureColor;

        drawArgs.device.VertexFormat = CustomVertex.PositionColoredTextured.Format;

        drawArgs.device.DrawIndexedUserPrimitives(PrimitiveType.TriangleList, 0,
            _ImageVertices.Length, _ImageIndices.Length / 3, _ImageIndices, true,
            _ImageVertices);

        drawArgs.device.Transform.World = drawArgs.WorldCamera.WorldMatrix;
    }
}
```

The functions displayed in the code above are basic 3D drawing functions and will not be explained here. Refer to a book on drawing with DirectX or at the following link. http://www.drunkenhyena.com/cgi-bin/dx9_net.pl

If you don't have any experience with 3D drawing with DirectX, this may not be a problem for you because World Wind already has a lot of different types of RenderableObjectLists and RenderableObjects. To draw paths, use the PathList and the PathLine or TerrainPath classes. To draw icons, use the Icons and Icon classes. To draw images, use the ImageLayer class. All of these classes and more can be found under the WorldWind.Renderable namespace. You can also create your own classes by simply inheriting the RenderableObject class and overriding the virtual functions. The RenderableObjectList class is written in a general enough form that it will probably suit your custom RenderableObject class. However, if you want something fancy, you could also inherit the RenderableObjectList class and override the functions you want to change.

Appendix B
Advanced Paparazzi Routine Development

The Paparazzi flight plan is organized very well. In addition to providing basic functions for waypoint navigation, it also allows the user to develop more advanced custom routines. This could be used for many things including takeoff, landing, surveying an area, etc. This is done by developing functions in C and including them in the flight plan using the call block.

B.1 Advanced Paparazzi Flight Plans

The Paparazzi flight plan is written with an xml file and defines the waypoints, sectors, blocks, and exceptions in the flight plan. The waypoints define different locations on the earth, the sectors are a collection of waypoints which define an area, the blocks are instructions given to the UAV, and the exceptions can redirect the autopilot to specific blocks depending on different situations. All of these are used by a preprocessor to generate C code which is then uploaded on to the autopilot. One file generated from the preprocessor is flight_plan.h which can be found in the paparazzi/var directory after compilation.

B.2 Incorporating Routines in Flight Plans

Custom routines can be used inside the flight plan structure using the call block.

```
<block name="Takeoff">
   <call fun="InitializeBungeeTakeoff(WP_Bungee)"/>
  <call fun="BungeeTakeoff()"/>
</block>
```

In the example above, if the Takeoff block is selected, first the initialization function will be executed with every iteration of the navigation loop until it returns a false value. Once it returns a false value, the autopilot will execute the BungeeTakeoff function until it also returns a false value. After the last function in the block returns a false value, the autopilot will execute the block after the Takeoff block. Here is the C code generated by the preprocessor from a flight plan including the Takeoff block above and a Standby block.

```
static inline void auto_nav(void) {
  switch (nav_block) {
    Block(0) // Takeoff
    ; // pre_call
    switch(nav_stage) {
      Stage(0)
        if (! (InitializeBungeeTakeoff(WP_Bungee)))
          NextStageAndBreak();
        break;
      Stage(1)
        if (! (BungeeTakeoff()))
          NextStageAndBreak();
        break;
      Stage(2)
        NextBlock();
        break;
    }
    ; // post_call
    break;

    Block(1) // Standby
    ; // pre_call
    switch(nav_stage) {
      Stage(0)
        NavVerticalAutoThrottleMode(RadOfDeg(0.000000));
        NavVerticalAltitudeMode(WaypointAlt(3), 0.);
        NavCircleWaypoint(3, nav_radius);
        break;
      Stage(1)
        NextBlock();
        break;
    }
    ; // post_call
    break;
  }
}
```

In addition to adding your functions in the flight plan, you will also have to include the header file at the top of the flight plan and the following code to the bottom of your airframe file.

```
ap.srcs += YourFile.c
sim.srcs += YourFile.c
```

B.3 Writing Custom Routines

All the navigation code goes in the sw/airborne/ directory in Paparazzi. You can add your function to an existing file or you can create your own. If you plan on contributing back to the Paparazzi community by posting your code online, the best option would be to create your own file. For each routine you want to create, you will want two functions: an initialization function and a main function. The code below shows how these functions for the bungee routine is defined in the header file.

```c
#include "std.h"
#include "nav.h"
#include "estimator.h"
#include "autopilot.h"
#include "flight_plan.h"

extern bool_t InitializeBungeeTakeoff(uint8_t BungeeWP);
extern bool_t BungeeTakeoff(void);
```

The functions are declared with the extern keyword so they can be used in the scope of the flight plan. They also need to return a boolean variable. The header files included in this example are important to include to give the routine access to variables within the autopilot and the flight plan.

In the source code file, the initialization and main function typically have the following format.

```c
#include "TheHeaderFile.h"

//Variables you want to use in both functions are declared here
enum TakeoffStatus { Launch, Throttle, Finished };
static enum TakeoffStatus CTakeoffStatus;
static uint8_t BungeeWaypoint;
static float initialx;
static float initialy;

bool_t InitializeBungeeTakeoff(uint8_t BungeeWP)
{
   //Initialize variables
   initialx = estimator_x;
   initialy = estimator_y;

   BungeeWaypoint = BungeeWP;

   float Currentx = initialx -(waypoints[BungeeWaypoint].x);
   float Currenty = initialy -(waypoints[BungeeWaypoint].y);

   /*
   Other stuff not mentioned here
   */

   //Initialize state machine
   CTakeoffStatus = Launch;

   return FALSE; //Return false if you want to move on in the flight plan
}

bool_t BungeeTakeoff(void)
{
   //State machine
   switch(CTakeoffStatus)
   {
   case Launch:
      //Follow Launch Line
      NavVerticalAutoThrottleMode(0);
     NavVerticalAltitudeMode(BungeeAlt+Takeoff_Height, 0.);
      nav_route_xy(initialx,initialy,throttlePx,throttlePy);

      kill_throttle = 1;

      //Find out if UAV has crossed the line
      if(AboveLine != CurrentAboveLine && estimator_hspeed_mod > Takeoff_MinSpeed)
      {
         CTakeoffStatus = Throttle;
         kill_throttle = 0;
         nav_init_stage();
      }
      break;
   case Throttle:
      //Follow Launch Line
      NavVerticalAutoThrottleMode(0);
     NavVerticalAltitudeMode(BungeeAlt+Takeoff_Height, 0.);
      nav_route_xy(initialx,initialy,throttlePx,throttlePy);
      kill_throttle = 0;

      if((estimator_z > BungeeAlt+Takeoff_Height-10) && (estimator_hspeed_mod >
          Takeoff_Speed))
      {
         CTakeoffStatus = Finished;
         return FALSE;
      }
      else
      {
         return TRUE;
      }
      break;
   default:
      break;
   }
   return TRUE; //Return true if you don't want to move on in the flight plan
}
```

The basic purpose of the initialization function is to initialize important variables and the state machine. In the example above, the initialx and initialy variables are set to the estimator_x and estimator_y variables. The estimator_x and estimator_y are defined in the estimator.h file and are the current x and y position of the UAV in the navigation coordinate system. There is also an estimator_z variable for the altitude of the UAV above sea level. The example of the initialization function also shows how to use waypoints in your routine. The waypoints declared in the flight plan are compiled in an array called waypoints. The elements in the waypoints array are structures with fields x, y, and a. x and y represent the position of the waypoint in navigation coordinates and the a field is the altitude of the waypoint above sea level. This array can be used in your routine if flight_plan.h is included in your header file. Also, when passing a waypoint to your function, the index to the waypoint in the array is passed.

The main function in the routine is basically a state machine. In the example above, the initialization function sets up the state machine to start in the Launch state. In this state, the autopilot is told to follow a line defined from the initial position of the UAV, when the initialization function was executed, to the position of the bungee staked into the ground. Once the UAV is hooked up to the bungee, stretched back and released, it will follow this line and ascend to the altitude defined in the NavVerticalAltitudeMode function. In addition, the kill_throttle variable (defined in the autopilot.h file) is set as one. This will make sure the propeller does not turn on and get tangled in the bungee or injure the person launching the UAV. The condition at the end of the Launch state switches the current stage to Throttle, turns the throttle on and resets the navigation settings once the UAV crosses the bungee position and reaches the specified minimum speed. The Throttle state navigates the UAV in the same way as the Launch state, except the throttle remains on. Once the UAV reaches the specified height and speed, the state will be switched to Finished and the function returns false. Since the function returns false at this point, the autopilot will move on to the next block.

B.4 Extra Features

To increase the flexibility of your routine you can include external variables which can be used with exceptions in your flight plan. For example, the following PolySurvey routine will allow the UAV to automatically survey any given convex polygon given the sweep width and waypoints which define the corners of the polygon. The UAV will continue to survey the area until the GCS operator tells it to move on to another block or until the exception in the first line is met. The exception tells the UAV to execute the Standby block when PolySurveySweepNum is greater than or equal to two. The variable PolySurveySweepNum is incremented every time the UAV sweeps across the polygon. Therefore after two sweeps, the UAV will deroute to Standby.

```
block name="Poly Survey">
   <exception cond="PolySurveySweepNum >= 2" deroute="Standby"/>
   <call fun="InitializePolygonSurvey(WP_S1, 5, 200, 45)"/>
   <call fun="PolygonSurvey()"/>
</block>
```

Along with the functions in your routine, to use a variable in the flight plan you only need to declare it as an external variable in your header file.

```
extern bool_t InitializePolygonSurvey(uint8_t FirstWP, uint8_t Size, float Sweep,
    float Orientation);
extern bool_t PolygonSurvey(void);
extern uint16_t PolySurveySweepNum;
```

Another useful feature is the ability to declare variables in your airframe file. This is helpful because it allows the user to tune a routine specific for each aircraft. The xml code below shows an example of how the variables are added for the BungeeTakeoff routine.

```
<section name="Takeoff" prefix="Takeoff_">
  <define name="Height" value="50" unit="m"/>
  <define name="Speed" value="10" unit="m/s"/>
  <define name="Distance" value="3" unit="m"/>
   <define name="MinSpeed" value="5" unit="m/s"/>
</section>
```

All the variables are declared in a section. Not only does this nicely organize the variables but it also gives them a unique name. This is done by using the prefix attribute of the section and appending it to the beginning of each of the variable names. In the example above, the variable Height would actually be defined as Takeoff_Height. Once the variables are declared in the flight plan, they can be used in your routine without adding any more code. Here is an example using some of the variables above.

```
bool_t BungeeTakeoff(void)
{
   //State machine
   switch(CTakeoffStatus)
```

```
  {
  case Launch:
     //Follow Launch Line
     NavVerticalAutoThrottleMode(0);
    NavVerticalAltitudeMode(BungeeAlt+Takeoff_Height, 0.);
     nav_route_xy(initialx,initialy,throttlePx,throttlePy);

     kill_throttle = 1;

     //Find out if UAV has crossed the line
     if(AboveLine != CurrentAboveLine && estimator_hspeed_mod > Takeoff_MinSpeed)
     {
        CTakeoffStatus = Throttle;
        kill_throttle = 0;
        nav_init_stage();
     }
     break;
  case Throttle:
     //Follow Launch Line
     NavVerticalAutoThrottleMode(0);
    NavVerticalAltitudeMode(BungeeAlt+Takeoff_Height, 0.);
     nav_route_xy(initialx,initialy,throttlePx,throttlePy);
     kill_throttle = 0;

     if((estimator_z > BungeeAlt+Takeoff_Height-10) && (estimator_hspeed_mod >
         Takeoff_Speed))
     {
        CTakeoffStatus = Finished;
        return FALSE;
     }
     else
     {
        return TRUE;
     }
     break;
  default:
     break;
  }
  return TRUE; //Return true if you don't want to move on in the flight plan
}
```