5-2005

Synchronized Line-Scan LIDAR/EO Imager for Creating 3D Images of Dynamic Scenes: Prototype II

Donald C. Anderton
Utah State University

SYNCHRONIZED LINE-SCAN LIDAR/EO IMAGER FOR CREATING 3D IMAGES OF DYNAMIC SCENES: PROTOTYPE II

by

Donald C. Anderton

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

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Robert T. Pack Major Professor	Dr. Scott E. Budge Committee Member
Paul Israelsen Committee Member	Dr. Rees Fullmer Committee Member

UTAH STATE UNIVERSITY
Logan, Utah

2005

ABSTRACT

Synchronized Line-Scan Lidar/EO Imager for

Creating 3D Images of Dynamic Scenes: Prototype II

by

Donald C. Anderton, Master of Science

Utah State University, 2005

Major Professor: Dr. Robert T. Pack
Department: Electrical and Computer Engineering

A second prototype integrated lidar/Electro Optical (EO) camera, or 3D Texel™ camera, has been developed by Utah State University (USU) Center for Advanced Imaging Ladar (CAIL) that collocates, both temporally and spatially, a CMOS digital camera readout with a time-of-flight pulsed lidar[1]. The first prototype uses a flying spot lidar with a double gimbal whereas this prototype uses a single gimbal and rotating mirror. The selection and design of the second prototype hardware components: lidar, EO camera, computer, synchronization box, power source (AC to DC and battery box) and battery charger is discussed and compared with the first prototype.

The new software used for scanning and tiling called LDScanner™ and

---

[1]The methods disclosed within this paper constitute USU owned IP protected by U.S. Patent 6,664,529, "3D Multispectral Ladar".

LDImager™, developed in conjunction with RappidMapper Inc., Salt Lake City Utah, is explained. Based on the operator input, LDScanner™ initializes the hardware components, sets them up for scanning, conducts the scan and analyzes the data files. LDImager™ takes the lidar and EO camera files and converts them into image files for 3D viewing. LDImager™ also allows the operator to analyze preprocessed data files from the 3D Texel™ camera and create a histogram of the processed image.

The EO requirements to control the region of interest (ROI), set camera properties, trigger the camera and set the file format are discussed. Third party toolkits including the EO camera manufacturer's demo program and Carnegie Melon University's (CMU) demo program were considered to implement EO requirements. The requirements ended up being implemented using a modified CMU demo program. The software created in the modified CMU demo program and the CMU1394.dll functions were made accessible through calls to the Camera.dll wrapper. The functions exposed through the wrapper were also used to construct a Camera Image dialog box. This dialog box displays the real time EO image and histogram. It also allows the operator to adjust the shutter speed, gain, brightness, white balance and select the scan azimuth extents.

(147 pages)

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A second prototype integrated lidar/EO camera, or 3D Texel™ camera, has been developed by Utah State University (USU) Center for Advanced Imaging Ladar (CAIL). The 3D Texel™ camera "synchronizes and aligns CMOS digital camera readouts with the scan motion of a time-of-flight pulsed lidar" [1]. This technique is CAIL proprietary intellectual property (IP) and has been proven by the development of two prototype 3D Texel™ cameras.

The 3D Texel™ camera technology rests on the blending of the output of a color digital camera with the output of a lidar. Digital cameras produce 2D x-y texel elements, or texels, without absolute or relative range information. Lidar provides absolute (with GPS) or relative 3D x-y-z range information with sub-centimeter accuracy. When these two data sets are combined, a high resolution 3D image with accurate range information is obtained. The combination of the lidar and digital camera sensors into a single camera is what CAIL refers to as a 3D Texel™ camera.

The blending of 3D and texture data sets is commonly referred to as data fusion. Data fusion is not a new idea, but the method that CAIL uses to perform data fusion is. The CAIL 3D Texel™ technology is unique in that the digital camera and lidar are combined such that each subframe of the image and pixel of lidar information is collected at the same time. This means that the output from the digital camera and lidar are collocated both temporally and spatially. By collecting the data in this manner, the 3D Texel™ camera can instantaneously create error-free three dimensional color images of

moving objects from a dynamic point of view.

Once collected, the lidar x-y-z point cloud is used to produce a wire mesh. The 2D digital camera output is then "painted" onto the 3D wire mesh to provide the texture information. The combination of the two data sets produces the final high resolution textured 3D image. The development sequence of the 3D image is demonstrated in fig. 1.1.

This 3D image information is represented on a computer screen in 2D. Although these images appear 2D, they can be rotated to viewpoints that are different from the viewpoint where the image was actually taken. This means that the 3D image viewer can show the view from an angle, not seen in person, as if a photo had been taken from that direction. Several different perspectives of the same image are shown in fig. 1.2.

There are many benefits to the CAIL method of producing 3D images. The system acquires 3D images instantaneously. This means that the images can be viewed and analyzed in real-time. Because the digital camera and lidar data is collected at the same time, the system can handle camera motion and/or object motion. Finally, the acquired data can be positioned accurately in geographic coordinates by using a GPS Georeferencing system.

The focus of this paper is the work done on this project by the author. This includes team effort as well as exclusive work. The author worked within the team on the hardware design, presented in Chapter 3, sections B, D, E and F, and the software design, presented in Chapter 4. The author worked exclusively on EO camera interfacing, presented in Chapter 5. Additional work, not completed by the author, is

included within Chapter 1, Chapter 2 and Chapter 3, sections A and C, to keep the author's work in context with the overall effort.

Chapter 1 covers the background of the first 3D Texel™ camera technology, Prototype I, developed by CAIL. This prototype was developed by Dr. Robert Pack, Paul Israelsen, Brandon Withers and Kylee Sealy.

Chapter 2 is the design overview of the second prototype, or Prototype II. The overall design of Prototype II was performed by Dr. Robert Pack and Paul Israelsen of CAIL. The design consisted of both hardware and software components.

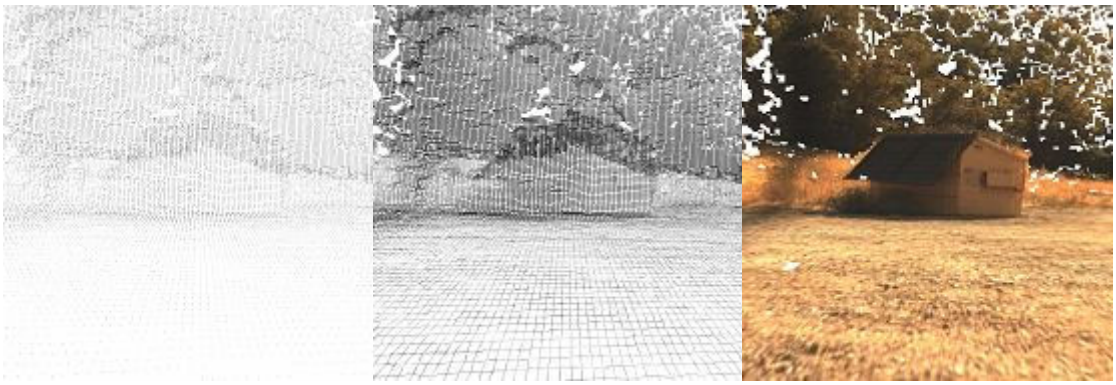

Fig. 1.1. Lidar x-y-z point cloud, wire mesh and textured 3d image.

Fig. 1.2. Original perspective, rotated top-view, rotated side-view.

Chapter 3 explains the specific hardware used in Prototype II and compares it to Prototype I. The lidar, EO camera and laptop were selected by Dr. Robert Pack and Paul Israelsen. The Synchronization Box was designed by Kylee Sealy and Shayne Rich. The Synchronization Box was assembled, tested and debugged by Kylee Sealy, Shayne Rich and the author. The Battery box and charger were designed by the author and Randy Christensen and then assembled and tested by Shayne Rich and the author. The power supply was selected by the author and Kylee Sealy. All cabling was assembled by Shayne Rich.

Chapter 4 presents the custom software LDScanner™ and LDImager™. The scanning flow, tiling algorithm and pseudo code were defined by Dr. Robert Pack. The tiling algorithm and pseudo code were coded into a C++ dll by Stan Coleby of Salt Lake City- based RappidMapper Inc. LDModeler™ was entirely developed by RappidMapper Inc. The LDScanner™ and the LDImager™ GUI's and associated control algorithms were developed and programmed by the author and Kylee Sealy.

Chapter 5 explains how the EO camera was interfaced by the author with LDScanner™. The LDScanner™ work was split into EO camera interfacing, lidar interfacing and GUI development. The author interfaced the EO camera, Kylee Sealy interfaced the lidar and both the author and Kylee Sealy assembled the GUI.

Chapter 6 is the design review and conclusion. Prototype II is summarized and the design review is discussed. The design review was completed by Dr. Robert Pack, Paul Israelsen, Shayne Rich, Kylee Sealy, Randy Christensen and the author.

A. Background

CAIL personnel invented the 3D Texel™ camera technology. The CAIL 3D Texel™ camera uses an integrated lidar scanner and a high resolution CMOS Electro Optical (EO) camera that are collocated both temporally and spatially to produce a 3D image. The lidar scanner produces a 3D x-y-z point cloud of the scene and the EO imager produces a high resolution image patch for each x-y-z point. The EO data set contains multiple patches that are then mosaiked to build up a 3D image composed of 3D texture elements (texels). The mosaiked Texel image can be viewed as a 3D image in any standard 3D viewer, such as LDModeler™ [1].

In previous work CAIL produced a prototype 3D Texel™ camera. Prototype I consists of hardware and custom software. The hardware components are the sensors used to collect the 3D x-y-z data and the image information. The software is used to control the data collection process, tile the collected data and view the final image. The hardware and software used in Prototype I will be discussed in this section.

B. Hardware

The Prototype I 3D Texel™ camera developed by CAIL consists of a single-channel flying-spot lidar, EO camera, a desktop computer with wireless monitor, synchronization box, battery pack and a battery charger. Figure 1.3 shows a simple schematic of where each of these components fit within the 3D Texel™ camera structure. In the following section the function of each piece of hardware is explained in further detail.

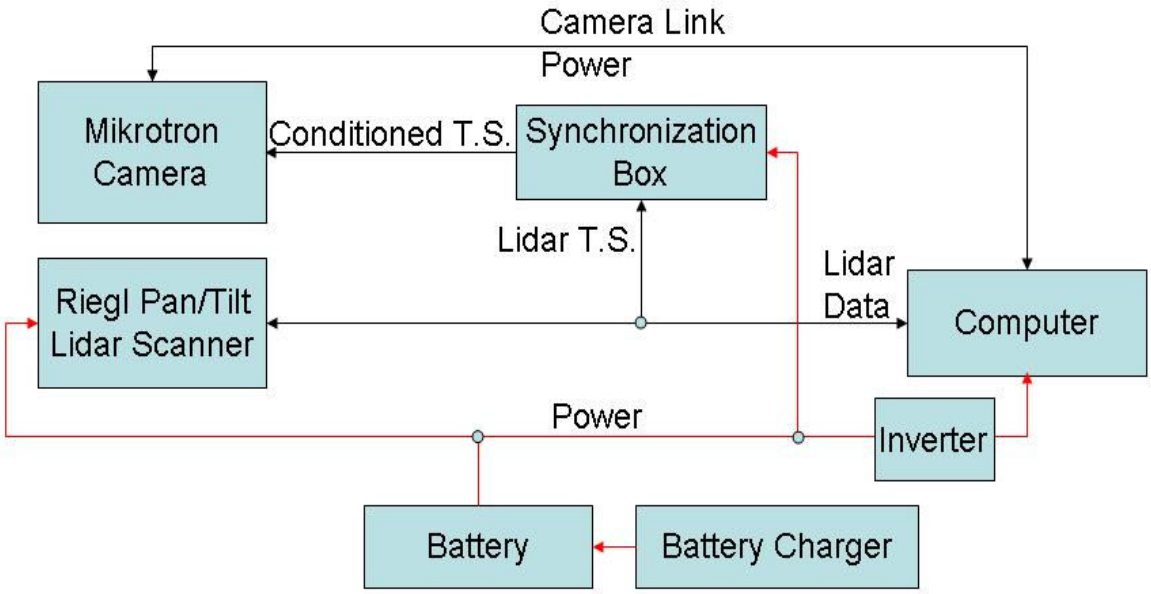

Fig. 1.3. Prototype I hardware diagram.

1) *Lidar:* Lidar (LIght Detection And Ranging or Laser Imaging Detection And Ranging) is a technology that determines distance to an object or surface using laser pulses. It is synonymous with the term LADAR (LAser Detection And Ranging) which is mainly used within military circles. The term "laser radar" is also in use, but is somewhat misleading as laser light, not radiowaves, is used. However, like the similar radar technology, the range to an object is determined by measuring the time delay between transmission of a pulse and detection of the signal reflected from a target [2].

The 3D Texel™ camera uses the lidar sensor to gather 3D information and create an x-y-z point cloud. The point cloud and EO data are aligned in hardware to produce a 3D Texel™ image. The lidar used in Prototype I is a Riegl LPM i300VHS with a pan/tilt unit for horizontal and vertical motion, fig. 1.4 [3]. The lidar uses a time-of-flight pulsed laser transceiver with a range accuracy of ± 5 cm (1-sigma) and a maximum shot rate of

1000 Hz. The beam divergence of this transceiver is approximately 3 mrad, which is equivalent to a laser spot size of 3 cm at a distance of 10 m. The spot size increases linearly with the distance from target up to a maximum of 400 m with objects of 80% or higher reflectivity. The maximum range at 20% reflectivity is approximately 180 m [1].

The laser rangefinder is attached to the Riegl pan and tilt mechanism. The pan and tilt mechanism uses a rotating table for horizontal movement with a maximum horizontal scan rate of 628 mrad/s (36^{o}/s). An oscillating arm is used for vertical movement with a maximum vertical scan rate of 1414 mrad/s (81^{o}/s). The pan and tilt unit has a pointing accuracy of approximately 300 μrad (0.018^{o}). At a shot spacing of three mrad intervals, the scan rate is limited to approximately 470 shots/s due to the maximum vertical scan rate [1].

2) Electro Optical Camera: The EO camera, commonly known as a digital camera, "reads out a 13 by 13 patch of RGB pixels within the subtended angle of a single lidar beam footprint" [1]. These images are tiled together to create a high resolution digital image of the scene. The EO camera used is a Mikrotron MC1303 high speed color CMOS digital camera with a focal plain array size of 1280(h) x 1024(v) pixels. The pixels are 12 x 12 μm in size with a fill factor of 40%. The camera is capable of up to 100 frames/s at full resolution and 5000 frames/s with a 100 x 100 pixel sub-frame or region of interest (ROI). Using the desired 13 by 13 patch, the camera produces pixels at approximately 170 times the shot rate of the lidar or 79000 pixels/s. The Mikrotron camera uses Camera Link™ interface, which requires an external PCI frame grabber, for data transfer at a rate of greater than 528 Mbs [1].

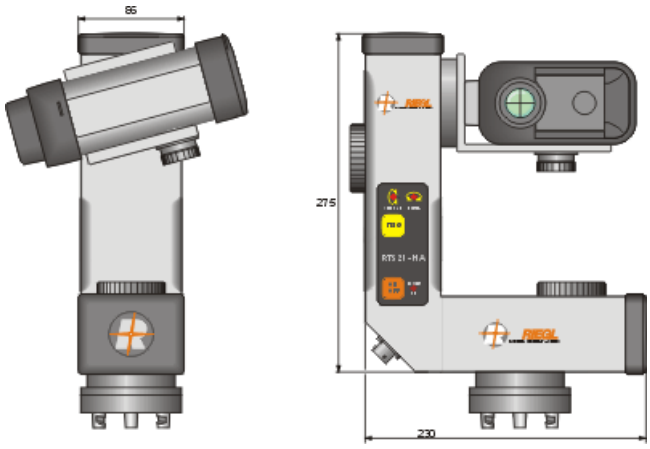

Fig. 1.4. Reigl pan/tilt scanner.

3) Inverter: The desktop computer requires a 120 V AC 60 Hz power supply. This means the computer can be plugged directly into a convenience outlet to run. However, the 3D Texel™ camera needs to be able to operate entirely on lead acid batteries. In order to use the computer in this configuration a power inverter is used to convert the DC battery voltage to AC voltage for the computer.

4) *Computer:* The computer controls data collection from the lidar and EO camera, the tiling of the EO images and can be used to view the 3D images. This means that the computer must be able to gather and handle large amounts of data quickly. The computer used in Prototype I is a custom built miniature desktop from Logisys Corporation. It has a 2.8 GHz (400 MHz FSB) main processor, 60 GB-7200 rpm hard drive, 1 GB pc 2700 333 MHz RAM, integrated Intel Extreme Graphic video card and CD-RW disk drive. The I/O ports included on the computer are serial DB9, parallel, Ethernet, CF card reader and USB 1.0. The desktop form factor was chosen, rather than a laptop, because the EO camera required a PCI slot for the frame grabber.

5) *Synchronization Box:* The lidar scanner produces a trigger signal whenever a

lidar shot occurs. This signal is used by the lidar and EO camera to coordinate the image capture, but it cannot be used directly by the EO camera because the trigger signal coming from the lidar camera stays high for only 6 μs. The Mikrotron cannot resolve a pulse as short as 6 μs. To get the EO camera to trigger, the lidar trigger signal needed to be extended by using a retriggerable one-shot circuit. A synchronization box was designed to accept the 6 μs lidar signal and extend this signal to a 142 μs signal that the EO camera can resolve as shown in fig. 1.5.

6) Power Source – Battery: Prototype I uses more than 160W at peak usage and must be able to operate for at least four hours in the field. The least expensive battery option that meets this power requirement is sealed lead-acid batteries.

Three PS-12180 Power-Sonic12V/18Ah batteries [4] were secured to a support structure and wired in parallel to produce a total of 54Ah. This provides approximately four hours of operation at the maximum discharge rate of 160W. Each of these batteries weighs approximately 12 lbs so the total battery weight is more than 37 lbs. This significantly increases the weight of the 3D Texel™ total field-portable camera package.

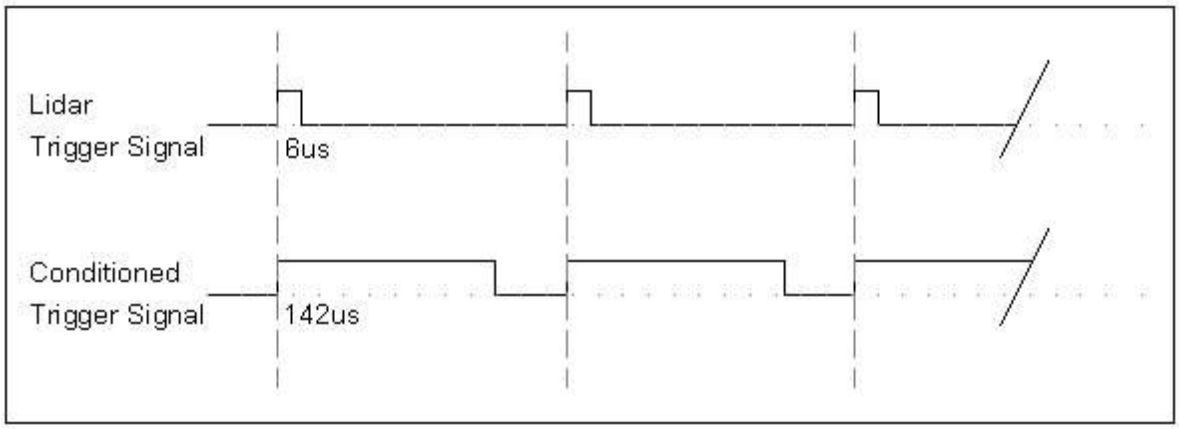

Fig. 1.5. Prototype I synchronization box signals.

7) Battery Charger: There are many manufacturers of lead-acid battery chargers. The one used for Prototype I is a Husky automatic on/off 12V lead-acid battery charger. The Husky battery charger provides a selectable 2A trickle charge or 6A high rate charge. The charger can be plugged into any standard 120V AC convenience outlet.

C. Software

The general structure of Prototype I control software is shown in fig. 1.6. The structure is very open which means there is no specific program flow mandated by the software in Prototype I. This means the operator has a lot of freedom but must have a good understanding of what the software does.

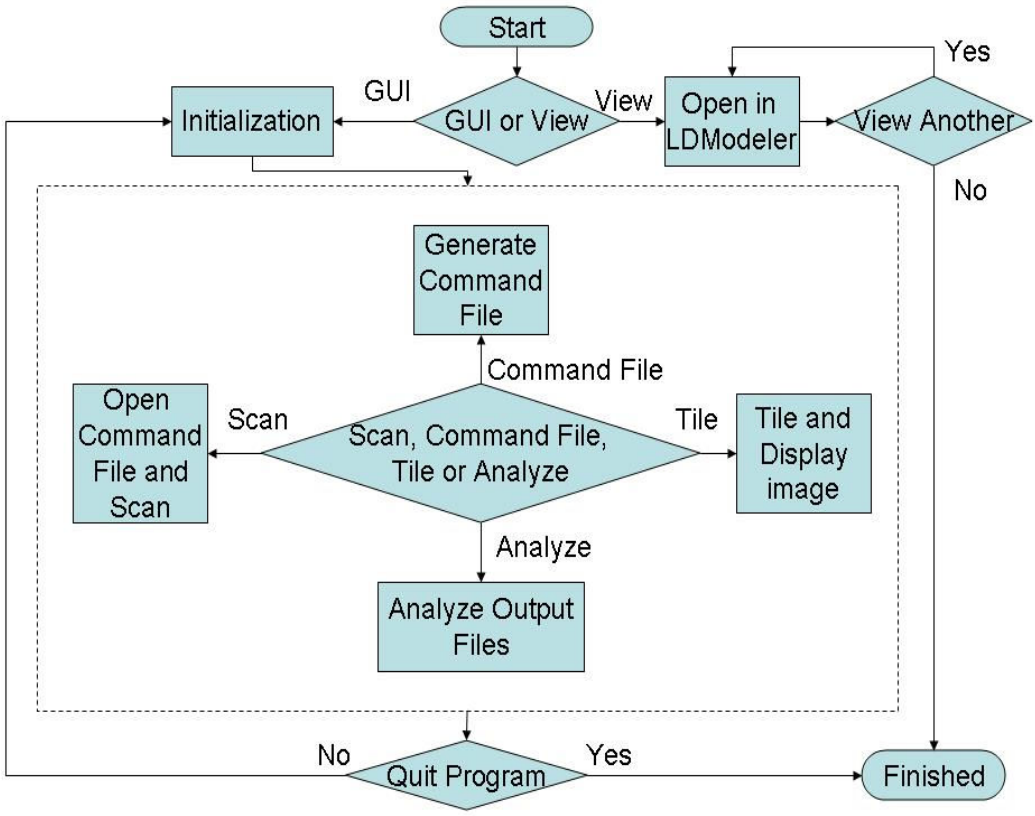

Fig. 1.6. Prototype I scan console structure.

When starting, the operator can choose to run the main graphical user interface (GUI) or view an image. If the operator chooses to run the GUI the program first initializes and then displays the main window. The main window has four options available, these options are to "Open Command File and Scan," "Generate Command File," "Tile and Display" and "Analyze and Histogram Files." Upon completion of any of the tasks the operator has the option to end the program or perform another task. If the operator wants to view the previously tiled images then LDModeler™, or any 3D viewing software, can be used for viewing [5].

Despite the open structure of the software, its main purpose is to scan, tile, analyze output files and view images. A more in-depth view of the operating flow is shown in fig. 1.7. This program flow diagram shows how each of the operator's main tasks are performed.

1) Performing a Scan: To conduct a scan, the operator first sets up the equipment at the desired location and then powers up the equipment. Once the equipment is running the GUI software is started on the computer. The GUI performs some basic initialization and then presents the operator with the four options discussed in the software introduction. The operator must first generate a command file.

When the Generate Command File option is selected, the GUI in fig. 1.8 is displayed. The operator can either use the joystick to move the 3D Texel™ camera to the desired scan limits or input them directly. If the operator uses the joystick, the opposite corners of the scan limits are designated by clicking the appropriate image icon, either upper left or lower right, when the 3D Texel™ camera is located at the desired start or

stop point. The operator also designates the shot spacing in the GUI. As this is done, an intensity histogram is displayed in real-time so the operator can adjust the camera's iris setting and other parameters such as gain. When all the parameters are set, the operator then saves the settings to a command file.

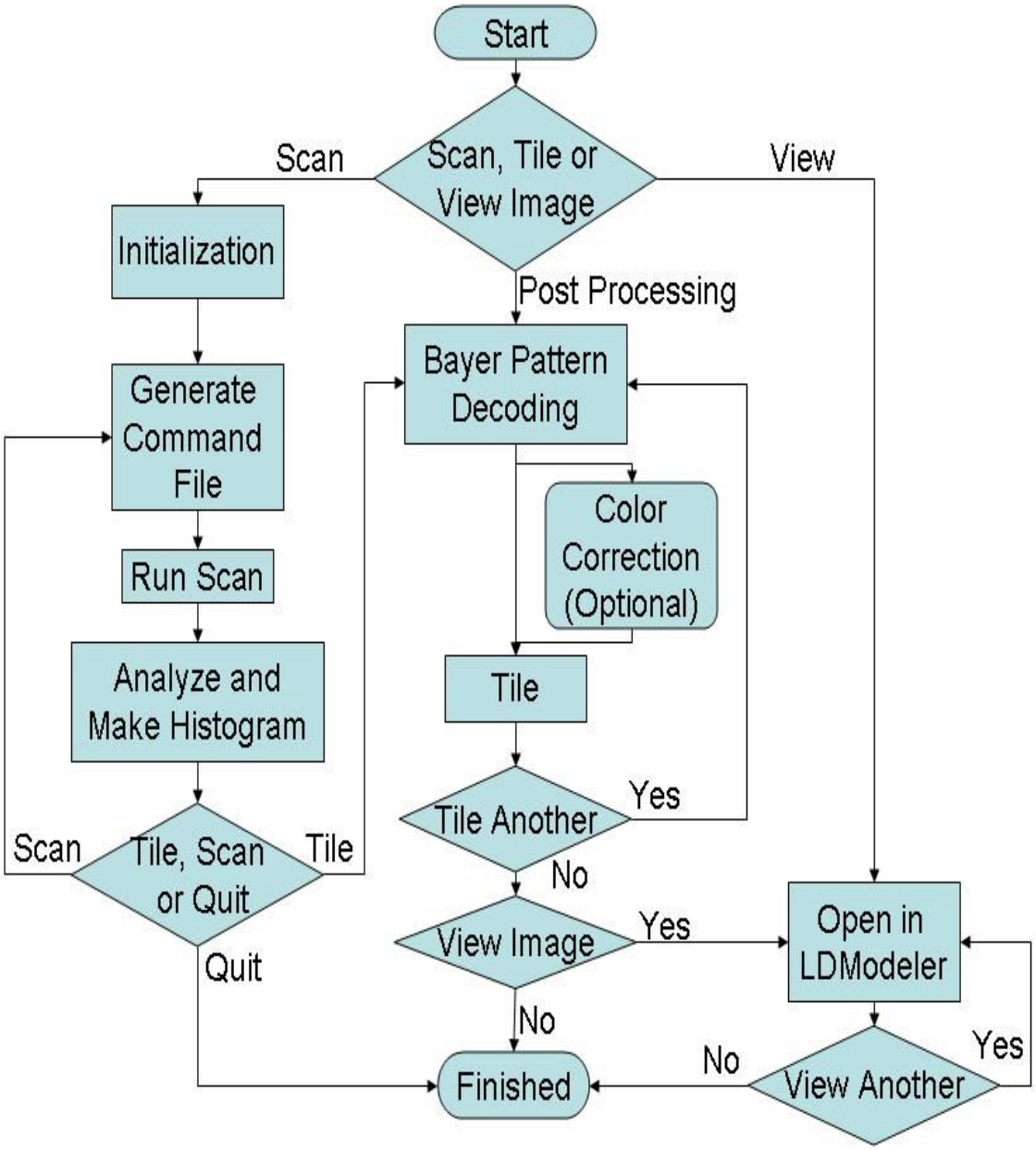

Fig. 1.7. Prototype I operating flow.

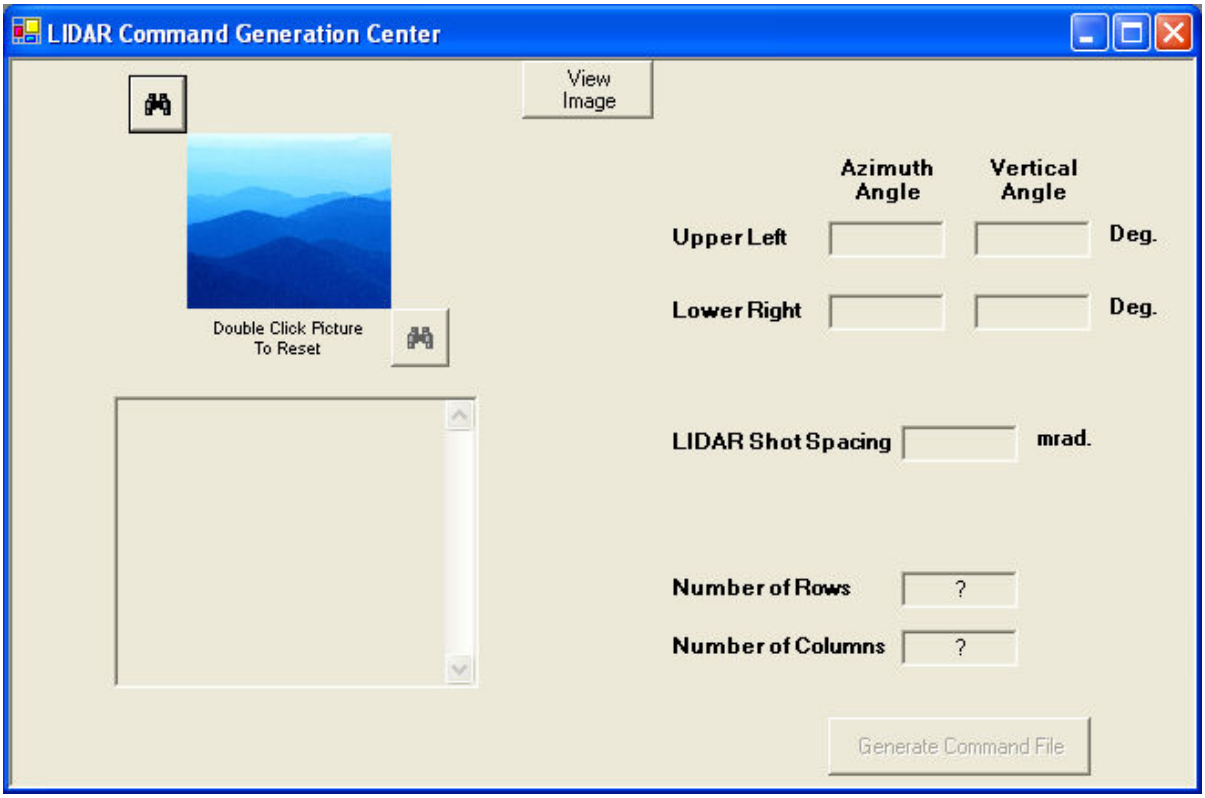

Fig. 1.8. Prototype I generate command file GUI.

Once the command file has been generated, the operator selects the Run Scan button. The operator is then prompted to select the command file that will be used to conduct the scan. The software loads the file and the lidar is configured using the parameters defined within the command file. When the scanning starts, the trigger signal from the lidar is used to synchronize the EO camera. The 3D Texel™ camera generates a lidar data file and an EO data file.

Upon completion of the scan, the operator selects the analyze option to create a histogram of the output images. The validity and quality of the images is determined by verifying that the file format and size are correct. At this point, the histogram of the image is an average intensity of all untiled EO frames in the file. Once the files are

verified, the scan process is complete and the operator can choose to conduct another scan, tile the frames into a texel image or quit.

2) Post Processing or Tiling: The 3D Texel™ camera generates two separate raw data files when conducting a scan. One of the files is a ladar (lad) file which contains the lidar point cloud. The other file is a custom texel (tex) file format developed by CAIL. This file gets its extension because it is a collection of all the texel images captured from the EO camera during scanning.

Post processing of the two raw data sets involves reformatting the files into a format that can be rendered by a graphics display engine. The image handling steps indicated in fig. 1.7 of Bayer pattern decoding, color correction and tiling is done automatically once the operator designates which files to use. In the Bayer pattern decoding step, the raw image information from the camera is converted into RGB format. A color correction filter is then applied to the RGB image so that the image reflects the true colors of the scene. The last step involves taking the RGB image frames from the previous steps and mosaicking them to produce a jpeg (jpg) texel image. This is called a texel image because each texture element can be mapped one-to-one with the lidar x-y-z data set [1]. When the post processing is complete, the operator can choose to process another data set, view the recently tiled data set in the 3D viewer or quit the program.

3) View: Once the lad and tex files have been produced, they can be rendered and viewed within the LDViewer/LDModeler graphics display engine. With some further reformatting, many 3D graphics engines are available which could be used to view these images.

D. Summary

With the development of Prototype I, the Center for Advanced Imaging Ladar proved that a lidar scanner and EO camera synchronized and calibrated within the same field-of-view can produce high resolution 3D images. The first prototype, or Prototype I, 3D Texel™ camera consists of hardware and software that generate a lidar x-y-z data set and raw image files. These data files are then used to generate a 3dd file and a single jpeg image that is matched to the lidar pixels. The x-y-z lidar file and tiled jpeg are then rendered and viewed as a 3D image in LDModeler™ from RappidMapper.

CHAPTER 2

DESIGN PROJECT OVERVIEW

The specific goal of the second prototype, or Prototype II, 3D Texel™ camera was to adapt to the use of a rotating mirror line-scan lidar and use a new EO camera with an on-board frame grabber and thereby improve the software design and features associated with the Scan, Tile and View tasks. An additional goal was is to improve the ease of use, speed and robustness of the overall design. This section gives a general overview of the design of Prototype II and explains the specific work performed on this project by the author as well as how the work was divided among CAIL and RappidMapper team members.

In contrast to Prototype I, the rotating mirror scanner allows a vertical line, or column, of pixels to be captured rapidly for every horizontal increment of the two-axis scan mechanism. This difference in scan mechanisms between the prototypes means that the method used to collect and then tile the texel images must be different. In Prototype I, each time a lidar range point was captured a 13 (h) x 13 (v) RGB pixel patch was collected from the EO camera. With Prototype II, each time a lidar column is captured, a 16 (h) x 2048 (v) RGB pixel column, or ribbon, is collected from the EO camera. This means that the tex file generated by Prototype II will contain hundreds of EO ribbons verses the thousands of EO patches produced by Prototype II. This also means that data collection in Prototype II from the lidar and EO camera happens more rapidly and a scan is completed more quickly.

A. Prototype II Hardware

As with the camera for Prototype I, the second prototype 3D Texel™ camera developed by CAIL consists of a lidar scanner, EO imager, computer, synchronization box, power supply and charger. Although the equipment is similar, the interaction of parts is different as shown in fig. 2.1.

The communication methods between the two prototypes are different. Tables 2.1 and 2.2 show the differences in communication from one device to another for each prototype. Each of the components within the system are listed twice; once in the left hand column, and once across the top row. These charts indicate one way communication from the device listed in the left hand column to the device listed in the top row. For example, the connection in Prototype II from "Synchronization Box" To "Lidar" is "Power and Trigger", whereas the connection in Prototype II from "Lidar" To "Synchronization Box" does not exist and is therefore grayed out.

In both prototypes the computer is used to setup, run and visualize the scans. Because the frame grabber is in the camera, Prototype II uses a laptop instead of the small desktop. This improved the power requirements because the computer could be powered on its own battery which eliminated the need for the power inverter. Because the power requirement changed, a smaller and lighter battery could be used to power the rest of the equipment. This affected the role of the power supply and required a custom built battery charger.

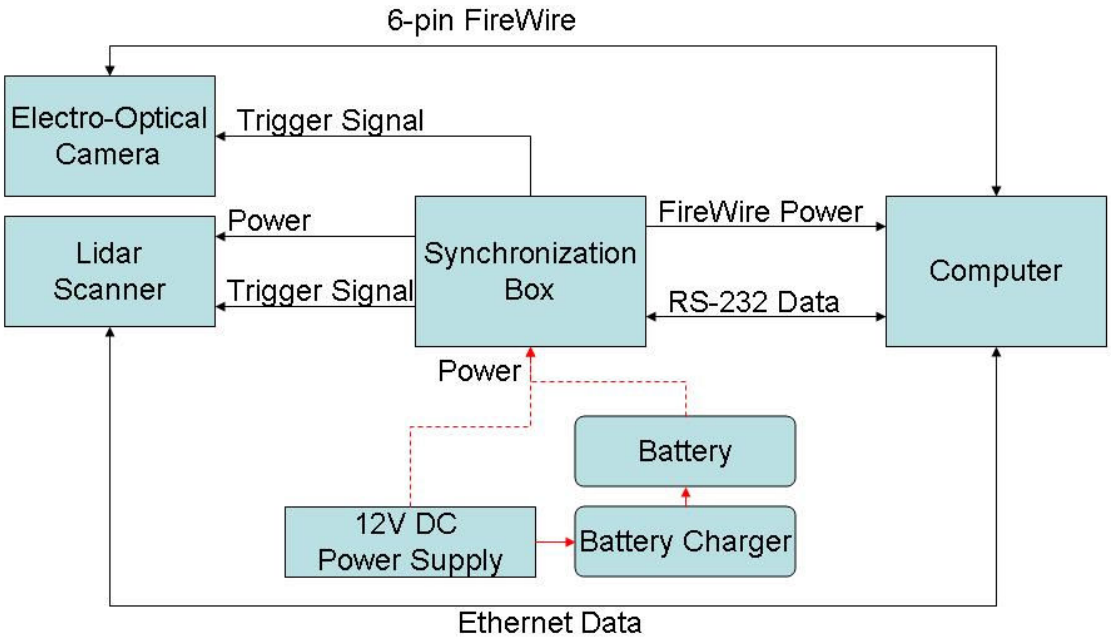

Fig. 2.1. Prototype II hardware relationships.

Another significant change is the role the lidar machine has in the timing process. In the first prototype, the lidar trigger signal was available externally so the EO camera could be triggered by the lidar. In Prototype II, the trigger signal is not available from the lidar so a Synchronization Box was designed to take over the timing responsibilities. Since the Synchronization Box was taking on a larger role with triggering, it was decided to add the power distribution into the Synchronization Box as well. Despite the hardware interaction differences between the two prototypes, they both achieve the same important objective of gathering lidar and EO data of the same scene at the same time.

The EO camera used in Prototype II is also different from the camera used in Prototype I. In the first prototype a Mikrotron EO camera with a Camera Link™ interface was used. In contrast, Prototype II uses an Imaging Solutions Group (ISG) EO camera with a FireWire™ interface. The Camera Link™ technology has a faster data

transfer rate than IEEE 1394a, or FireWire™, but does not have a universal standard like IIDC DCAM Specification Version 1.3 FireWire™ standard. "Camera Link™ does not have an associated standardized communication protocol such as that associated with FireWire™" [6]. Because of this an external capture card that is located in the computer is required.

The capture card was expensive and required a PCI slot. This limited the computer choice to a desktop and affected all aspects of the 3D Texel™ camera design. In the end, the PCI card has been the weakest part of the first prototype and has required service at regular intervals. Because of these downfalls, an EO camera with a standardized communications protocol and on-board frame grabbing was desired.

TABLE 2.1
PROTOTYPE I HARDWARE INTERFACING

To	Lidar	EO Camera	Computer	Synchronization Box	Battery	Charger	Power Supply
Lidar		Synchronization Box	Serial / Parallel	Trigger Signal			
EO Camera			Camera Link				
Computer	Serial / Parallel	Camera Link		Serial DB9			
Synchronization Box		Trigger Signal	Serial DB9				
Battery	2-Pin Power	Computer	2-Pin Power				
Charger					2-Pin Power		
Power Supply					Charger	2-Pin Power	

TABLE 2.2
PROTOTYPE II HARDWARE INTERFACING

To	Lidar	EO Camera	Computer	Synchronization Box	Battery	Charger	Power Supply
Lidar			Ethernet				
EO Camera			FireWire				
Computer	Ethernet	FireWire		Serial DB9			
Synchronization Box	Power and Trigger	Trigger	Serial DB9				
Battery	Synchronization Box	Synchronization Box		2-Pin Power			
Charger					Distributed Power		
Power Supply	Synchronization Box	Synchronization Box		2-Pin Power	Charger	2-Pin Power	

B. Prototype II Software

Based on experience with Prototype I, a program flow was established as shown in fig. 1.7. This program flow is straight forward and simple. Because of this, the program structure for Prototype II followed the program flow established with Prototype I. One can think of Prototype II more like a scan Wizard. The operator is prompted to answer simple questions which determine subsequent options as shown in fig. 2.2.

The program flow of Prototype I was divided into three main tasks: Scan, Tile and View. With Prototype II, these tasks were divided into three separate programs called LDScanner™, LDImager™ and LDModeler™. These programs simplify the scan process by automatically performing most tasks the operator performed manually in Prototype I. For example, with Prototype I, the Scan task required that the operator create a command file, conduct the scan and analyze the files in three separate manual tasks. In Prototype II all three of these tasks are accomplished automatically with minimal operator input. This is explained in further detail in the following sections in this chapter.

1) *Performing a Scan:* The scanning is accomplished through the LDScanner™ software. The scan portion of the 3D Texel™ flow is where the 3D data sets are collected. The operator chooses to conduct a scan once the hardware (lidar, EO camera, synchronization box and laptop) is set up at a desired location. The power is then turned on and the LDScanner™ software is started on the laptop. This software generates and saves a command file and then automatically uses the command file to configure the lidar for scanning. The EO camera properties are also configured at the same time. Once the

hardware is configured, LDScanner™ starts the scan through the Trigger box which controls the lidar and EO camera capture modes.

When the scan is finished, the operator can choose to tile the ribbons into a texel image, conduct another scan or quit the program. If the operator chooses to run the tiling program, LDScanner™ closes, the lidar is automatically parked in the storage position and LDImager™ opens. If the operator chooses not to run the tiling program, the console automatically recycles and the scan process is started over again. If the operator chooses to quit, then the lidar is automatically parked in the storage position and LDScanner™ closes.

2) *Post Processing or Tiling:* The tiling program, called LDImager™, uses a previously collected set of ribbon shaped frames to create a texel image that corresponds to points in the 3D point cloud. When using LDImager™, the operator opens the files for tiling and then runs the program. LDImager™ outputs a texel image in jpeg or other standard image format.

After tiling the data sets, the operator can tile another, quit the program or open the image or images in the viewer. The tiling algorithm and implementation will not be discussed in this paper, but the GUI to the tiling algorithm is discussed in Chapter 4, section B.

3) View: Like Prototype I, the 3D images that are collected and tiled are then viewed with LDModeler™ from RappidMapper. The tiling implementation will be discussed briefly in Chapter 4. After viewing an image the operator can choose to view another image or the program can be closed.

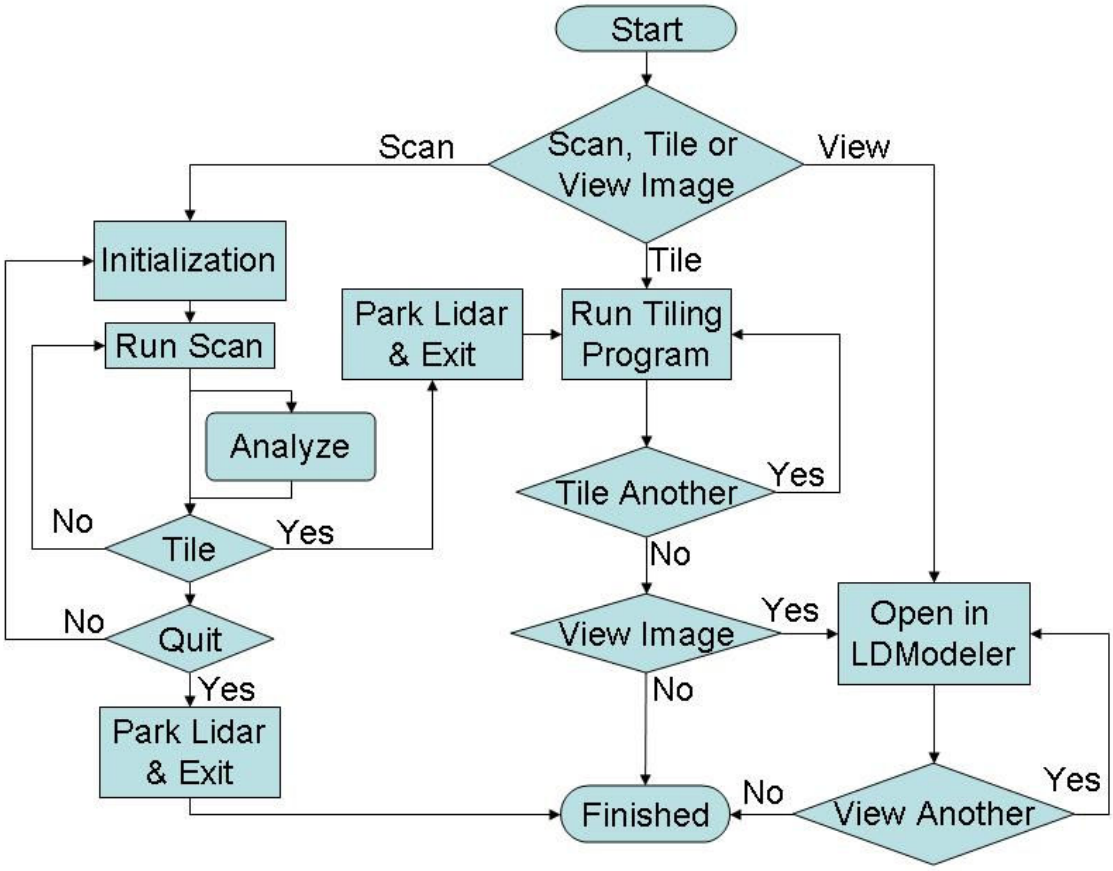

Fig. 2.2. Prototype II LDScanner™ program structure and program flow.

C. Design Project Summary

Many things were learned while constructing and testing the first prototype 3D Texel™ camera which affected the design of Prototype II. Using Prototype I as a model, CAIL designed and built Prototype II. This means that the hardware and software components are similar in both designs. These components will be discussed in the following sections.

CHAPTER 3

PROTOTYPE II HARDWARE

A 3D Texel™ camera consists of the following major components: lidar transceiver, two-axis scan mechanism, EO imager, computer, synchronization box, power supply and charger. An explanation of each of these components used in Prototype II is discussed and compared with the similar device used in Prototype I.

A. Lidar

The lidar used in Prototype II is the Riegl LMS-Z210i. This lidar consists of three main components that will be discussed. These components are the lidar transceiver, the two-axis scanning mechanism and the communications and power.

1) Lidar Transceiver: The laser transceiver is a time-of-flight pulsed laser with an accuracy of ± 15 mm (1-sigma) with a maximum laser pulse repetition rate of 24000 Hz and a mean measurement rate of 1/3 of PRR, or 8000 measurements/s for rotating line scan mode and line scan angle range approximately 80 degrees. The beam divergence of this transceiver is 3 mrad, which is equivalent to a laser spot size of 3 cm at a distance of 10 m and increases linearly with the distance from target up to a maximum of 400 m with objects of 80% or higher reflectivity. The maximum range at 20% reflectivity is approximately 180 m [7].

2) *Two-Axis Scanning Mechanism:* A major difference between the lidar scanners used in Prototype I and Prototype II is the method by which the two-axis

scanning is accomplished. The two-axis scan mechanism on Prototype II uses a rotating table for horizontal movement and a rotating mirror for vertical movement, fig. 3.1 [8].

The horizontal motion for Prototype II is accomplished by panning the laser transceiver on a rotating table. The range of motion is limited to 360^{o} by end switches and a mechanical end stop [9]. The maximum horizontal scan rate, using a shot spacing of 4 mrad/s and assuming 20 vertical lines per second, is 80 mrad/s (4.6^{o}/s). The shot spacing and azimuth limits significantly affect the horizontal scan rate and overall scan time.

The vertical scan mechanism of Prototype I is via nodding of the entire transceiver. On the other hand, Prototype II uses a three faceted rotating/oscillating mirror that records a line of data on each facet per rotation, or three lines per rotation, up to 20 line scans per second [9]. This mirror is mounted above a fixed transceiver and has a field of view (FOV) of 80^{o} in the vertical (elevation) and 360^{o} in azimuth.

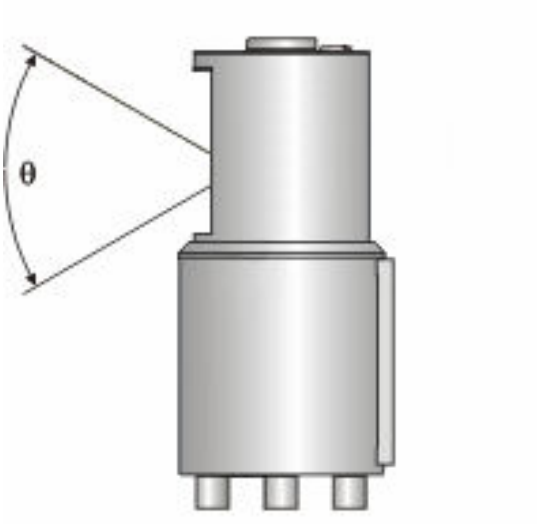

Fig. 3.1. Reigl rotating table/rotating mirror sensor.

The vertical capture rate is independent from the shot spacing and elevations selected. The lidar will always capture 20 vertical lines per second when the mirror is rotating at a relatively constant 400 rpm. By using a rotating mirror for the vertical scan mechanism, Prototype II can collect data an order of magnitude faster than Prototype I. A scan that took minutes with Prototype I can be scanned in seconds with Prototype II.

3) Communications and Power: The communication between the lidar and computer in Prototype II can be performed by either a serial and parallel cable combination or an Ethernet connection. The Ethernet connection was used for this communication to reduce the cabling and provide a simpler connection method. The triggering and power for the lidar are performed through the main lidar cable. According to the LMS-Z210i manual, the lidar requires a TTL voltage signal for triggering and a single DC power supply with a nominal output of 15V DC but can operate within the 12V to 28V DC range [9].

B. Electro Optical Camera

The EO camera selected needed to meet certain design requirements. These requirements are that the EO camera has a high resolution addressable CMOS array, an internal frame grabber, an external trigger capability, high speed data transfer and customization possibilities. An internal frame grabber enables the use of a laptop computer.

The Imaging Solutions Group (ISG) LW-3-S-1394-C “Smart” Digital Imaging Module meets these criteria. This EO camera has an internal frame grabber that interfaces with the computer using industry standard IEEE 1394a (FireWire™) and a

custom triggering cable. The power to the camera is supplied through the FireWire™ cable. The shutter is synchronous (rolling) to provide high speed capture rates. It can capture up to 12 frames / second at the maximum resolution of 2048 x 1536 Pixels (3.1 Mp), 27 frames / second at full frame (1.3 Mp), 90 frames / second at VGA quality. It can be tuned anywhere below or above these speeds based on the Region of Interest (ROI) designated and the shutter speed or exposure time. The exposure time is tunable through software [10,11].

Two different external triggering options are available, opto-isolated (up to 60 fps) and differential (in excess of 60 fps). The opto-isolated signal is a +5/0 V on/off signal. The differential triggering option requires a ± 5 V on/off TTL signal produced by a differential translator (RS-485 or RS-422) module. The camera includes a Multiple Frame Image Buffer (SRAM) for up to 10 color images. The camera is fully compliant with the IIDC DCAM Specification Version 1.3 [8, 9].

In addition to these standard features, ISG supports full customization of the imager with an on-board FPGA that allows the operator to customize configuration, image processing, color settings and I/O [9]. For this design these customizable features of the camera were not used.

C. Computer

The computer is used as the hub between all of the hardware components. Custom software was designed to interface the lidar, EO camera and synchronization box all on the computer. The computer must be able to collect and store large amounts of data in real time, process the large files to produce the tiled jpeg image and view the 3D

images. A computer with a Pentium M 1.5 GHz+ processor, 40Gb Hard Drive, 528 Mb+ of RAM and 128Mb+ dedicated video memory meets these requirements. In addition, the computer must have FireWire™ with 12V power, serial DB9, Ethernet, and a way to export data (cd/dvd burner, usb port for thumb drive, etc…). Finally, the computer must be compact, be able to operate in rugged conditions without problems and have enough battery power to operate in excess of four hours.

Desktops are much larger, heavier and require more power than a laptop computer. They are also not as robust since they are not built to be transportable. This was a major weakness in Prototype I, so, from the start, Prototype II was designed around a rugged laptop computer. The laptop chosen was a Panasonic Toughbook™ CF-73 provided by RappidMapper. This laptop met or exceeded all of the minimum requirements for operation.

D. Synchronization Box

The author did not design the printed circuit board and associated circuitry within the synchronization box but was very involved with the fabrication, testing and integration of this equipment into the overall system. Therefore, a detailed description of the function and relationship of the synchronization box within the system is included in this section.

The synchronization box used in the first prototype received the trigger signal from the lidar and then outputs a modified signal to the camera. The new synchronization box does much more. The synchronization box in Prototype II is designed to distribute power to all components in the 3D Texel™ camera except the

laptop. It also provides the trigger signal for the lidar and EO camera. The synchronization box requires a 13-18V DC power source. The input power is fed through the synchronization box directly to the lidar so the synchronization box operating voltage was selected to be within the lidar power supply range.

The synchronization box also supplies the EO camera with a 12V power supply fed back to the PCMCIA IEEE 1394 FireWire™ card located in the laptop computer. In addition, there are internal power requirements on the synchronization box. The optimal voltage of 14.8V is converted to 5V for the trigger signal to the lidar and then from 5V to 3.3V for the microcontroller supply voltage. The lidar trigger signal is fed through a buffer to a FET and then to the lidar. The FET on/off is controlled by the microcontroller.

The trigger signal from the synchronization box is generated by a Zilog Z8 Encore™ 40 pin microcontroller (part no. Z8F6421) programmable in C++ using the Zilog Developer Studio II (ZDS-II). This microcontroller provides two trigger signals, one for the lidar and one for the EO camera. The lidar trigger signal is a TTL step signal that starts the lidar on the rising edge as the signal goes high (+5V) and stops the lidar on the falling edge as the lidar goes low (0V) [12]. The EO camera trigger signal is a one-shot pulse width modulated +/- 5V differential signal that triggers the camera on the rising edge of the signal [10]. This signal is repeated for the duration of the scan at a frequency of approximately 60 Hz. The lidar receives the trigger signal buffered from the microcontroller and the EO camera receives the trigger signal buffered through a differential translator (RS-485) from the microcontroller. The lidar and trigger signals

generated for a single scan are shown in fig. 3.2. These signals will continue in the manner shown until the scan is complete; at which time both the lidar and EO camera trigger signals will go to 0V.

The synchronization box microcontroller communicates with the laptop through the RS-232 serial DB9 port. A switch inside the synchronization box turns the microcontroller from operating to programming mode so that the Zilog Z8 chip can be programmed in-line. The microcontroller is programmed to communicate with the laptop using the following commands: R-Response request (checks for communication), E-End (stops triggering and counter), C-Count (reports number of pulses sent) and B-Begin (resets and starts counter).

The outside of the synchronization box has the ports discussed earlier as well as an in-line fuse holder and heat sink on the side. The top of the synchronization box contains a voltage meter with on/off switch, a power-on LED (green-solid) with on/off switch and a CPU busy LED (yellow-blinking).

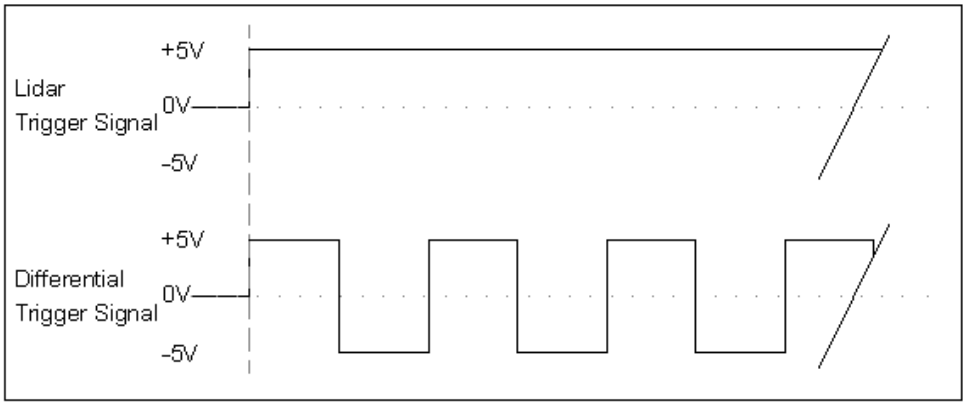

Fig. 3.2. Lidar and EO trigger signals.

E. Power Source

Power to the 3D Texel™ camera can be provided through a rechargeable battery box or a DC power supply. The power source for the 3D Texel™ camera was designed by the author around the lidar supply voltage of 13V-18V with a nominal voltage of 15V. The power source supplies power for all components in the system except the laptop. It connects directly to the synchronization box and power is distributed from there. The two different types of power sources used were the AC to DC 14V Power Supply and the lithium battery box.

1) AC to DC: A DC power supply was chosen that could run the 3D Texel™ camera from an AC outlet as well as run the battery charger or both at the same time. The power supply runs from a standard 120V outlet. The output voltage provided is 13.8 – 14.2V with a maximum current of 60A.

2) Battery Box: The battery is designed to be lightweight and able to power the machine for at least four hours. After reviewing available battery types and limitations, it was decided that Lithium Ion batteries would be used for the battery box because of their optimal series voltage of 14.8V and high-power density.

The total synchronization box power required for the Prototype II 3D Texel™ camera is the sum of the maximum power requirements of the lidar, EO camera and the synchronization box itself. The lidar requires approximately 50W, the EO Camera 3W and the synchronization box equipment approximately 2W, for a total of approximately 55W. The 3D Texel™ camera must be able to run at least four hours on battery power. The formula used to calculate the needed battery A-h is shown below where W=Watts

(55W), h=hours (4h), V=Voltage (12V) and Ah=Amp-hours

$$W * h = V * Ah.$$

$$Ah = \frac{W * h}{V}.$$

Using this formula, the battery box current rating needed to be at least 14.864 Ah. To allow for possible individual battery cell failure and extended life, the battery box current rating was designed to have 20 Ah. This will give a little more than five hours of run time at full discharge rate.

The battery box is made up of packs of batteries. Each pack has four 3.2V Li-Ion cells which combine to provide 2 Ah of current. These cells are placed in series to create a 14.8V, 2 Ah pack. Figure 3.3 shows a picture of one of these battery packs. The battery box contains ten of these packs wired in parallel to create a 14.8V, 20Ah battery pack.

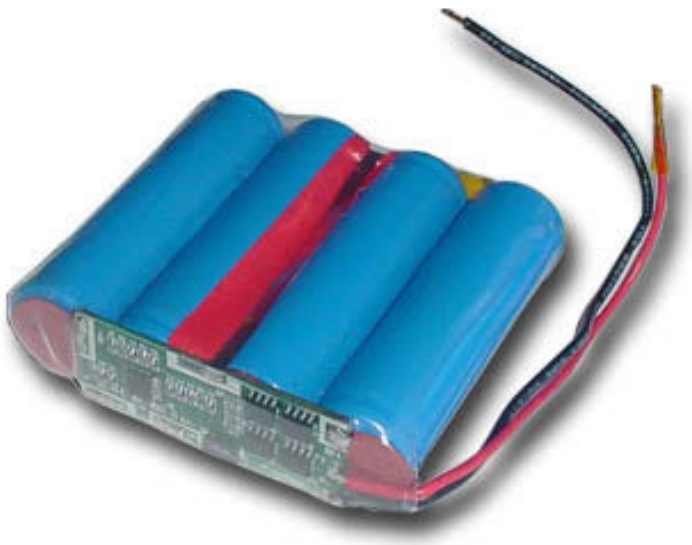

Fig. 3.3. Individual battery pack with protection circuit.

The battery packs can be charged at a maximum rate of 1C, at 2.4A/C or 2.4A, and can be discharged at a maximum rate of 10C, at 2.4A/C or 24A. The maximum discharge expected from the 3D Texel™ camera is 4.583A, so these batteries can provide the necessary power at maximum discharge rate. If the battery charge drops below 4.2A, it is no longer usable. The Cut-off voltage for the batteries is 12V. Each battery pack has a printed circuit board (PCB) with poly-switch to protect the batteries against charging, discharging, minimum charge and cut-off voltage limitations [13].

The battery pack required to run Prototype I weighed more than 37.8 lbs. In Prototype II, each pack in the battery box weighs 6.9 oz and there are ten packs so the final weight of the battery box is approximately 4.3 lbs. This is a significant improvement from Prototype I and greatly reduces the overall weight of the 3D Texel™ camera [13].

F. Battery Charger

The battery charger is required to charge the 14.8V, 20Ah battery box discussed earlier. Rather than using a single charger to charge all the batteries at once, which would take a considerable amount of time, the author suggested and designed the charger and battery such that multiple chargers could be used in parallel; this reduces the battery box charging time. To accomplish this, five chargers capable of charging two battery packs each were connected to the battery box such that two 14.8 V battery packs are connected to one battery charger.

The charger was constructed using Apache Li-poly Smart Charger 2500 for 1-4 lithium cells (3.2-14.8V) at a charge rate from 250mA to 2500mA shown in fig. 3.4. The

number of cells and charge rate are selected using jumper connections on each individual charger so they won't be bumped into different settings. Each charger has reverse voltage protection and number-of-cell protection circuits with LED indicators notifying the operator of problems and charging status. The charger LEDs were wired to the outside of the box so that all the chargers could be enclosed but still indicate the charging conditions.

The individual chargers get extremely hot when charging so the charge current was reduced to 750mA. In addition, a cooling fan attached to the outside of the box runs constantly when the charger box is powered. The charger connects to any 12-15V battery or power supply [14]. The completed charging box is shown in fig. 3.5.

G. Prototype II Hardware Summary

The second prototype 3D Texel™ camera created by CAIL incorporated hardware that was similar to the first prototype. A lidar using a line-scan two-axis scan mechanism with a standardized EO camera made up the main sensors in the system. Selection of these components allowed the desktop computer from Prototype I to be replaced by a robust laptop computer. This reduced the power requirements and weight of the whole system. In addition, the role of the Synchronization Box changed from signal conditioning in Prototype I to power distribution and timing coordination in Prototype II. Overall, the hardware chosen and designed in Prototype II resulted in a robust, much lighter weight and significantly faster 3D Texel™ camera.

Fig. 3.4. Apache li-poly smart charger 2500.

Fig. 3.5. Battery charger.

CHAPTER 4

PROTOTYPE II SOFTWARE

The hardware components are combined and coordinated using custom designed software developed and coded by the author and Kylee Sealy. There are three programs that were written to scan, tile and view the images. These programs are LDScanner™, LDImager™ and LDModeler™. LDScanner™ and LDImager™ were created by CAIL and LDModeler™ was created by RappidMapper. The software was designed on a Microsoft Windows XP platform using Microsoft Visual Studio .Net 2003. The software was written in Visual Basic and C++. This chapter describes the design of LDScanner™, including the Advanced Profile Editor, and LDImager™ software.

A. LDScanner™

The 3D Texel™ camera is controlled by the operator through the LDScanner™ software. As shown in fig. 4.1, LDScanner™ provides the communication interface between the lidar, EO camera and synchronization box. It was programmed in Visual Basic (VB) and C++. The VB portion contains the code that controls the lidar, the Camera Image dialog box and the graphical user interface (GUI). The EO camera controller and associated tiling algorithm were coded in C++.

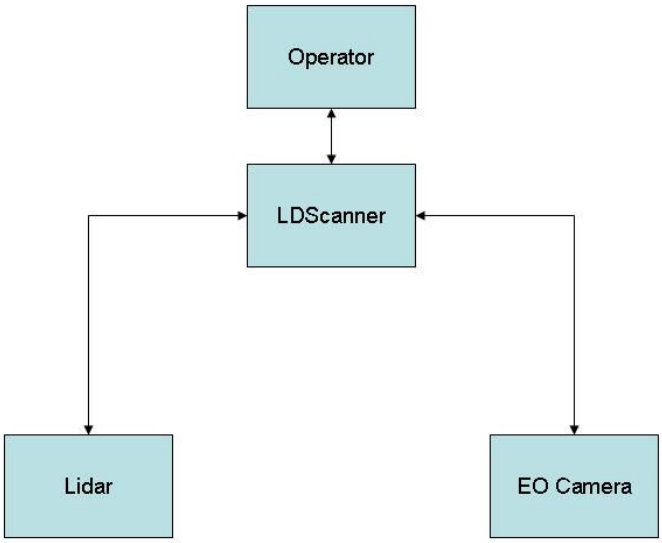

Fig. 4.1. LDScanner™ operator interface.

The reason why two languages were used is because VB allows a much more rapid development time. The original goal was to produce the entire software package in VB, including the EO camera controller. It was discovered that the EO camera interface between LDScanner™ and the EO camera could not be coded in VB. A considerable amount of development on the LDScanner™ software had already taken place by the time the EO camera interface limitations were discovered so, rather than starting over in C++, a combination of languages was used. This will be discussed in further detail in Chapter 5.

The basic program flow is shown in fig. 4.2. Each of the dashed line boxes indicate a major step that the software needs to complete before continuing. Each of these steps will be discussed in more detail following the brief overview. The program operates like a wizard in that each step prepares the operator for the next step in the process. As one step is completed the program moves to the next step.

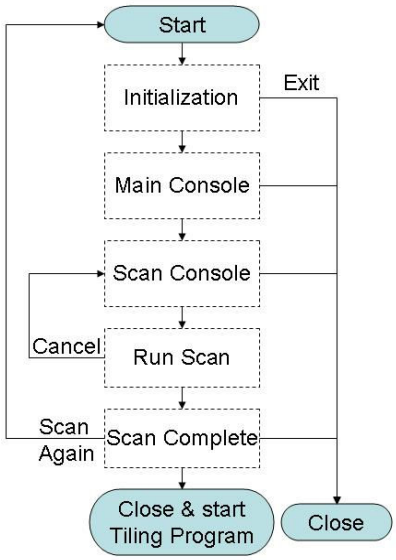

Fig. 4.2. LDScanner™ program flow.

The LDScanner™ GUI controls the flow of the program through the scan process. It consists of two main windows called Main Console and Scan Console. Both of these were designed to emulate a field computer shown in fig. 4.3. The consoles display different sub-screens as the operator progresses through the scan process. At any point during the program flow, except during scanning itself, the operator can close the program by clicking the power button in the lower right portion of the console graphic.

1) Initialization: When starting, LDScanner™ performs some basic initialization of the lidar, EO camera and the console. Often the Main Console opens before the initialization has completed in which case Initializing… is displayed in the sub-screen. If there are problems with initialization, the Main Console sub-screen displays the error as text and a button labeled "Connect again?". This allows the operator to correct the problem and then re-initialize LDScanner™ without having to restart the program.

Fig. 4.3. LDScanner™ console graphic.

The initialization process is shown in fig. 4.4. This process involves checking for proper communication between the computer and the three hardware components in the system. If the devices are not plugged in, not turned on or they have communication problems, LDScanner™ will detect the problem, notify the operator and will not proceed until the problem has been fixed. If all the hardware is operating properly then the program continues to the last step of the initialization.

The last initialization step is to check for the setup.xml file. This file contains the configuration parameters from the last scan performed. If the file exists, these parameters are read into the program and set as the default values. If the file does not exist,

LDScanner™ will use preset default values to generate a setup.xml file. Once the values from the setup.xml file have been loaded, the Main Console displays the lidar voltage and indicates the 3D Texel™ camera is initialized and ready to run. A button labeled Setup Scan prompts the operator to continue. The associated screen shot is shown in fig. 4.5.

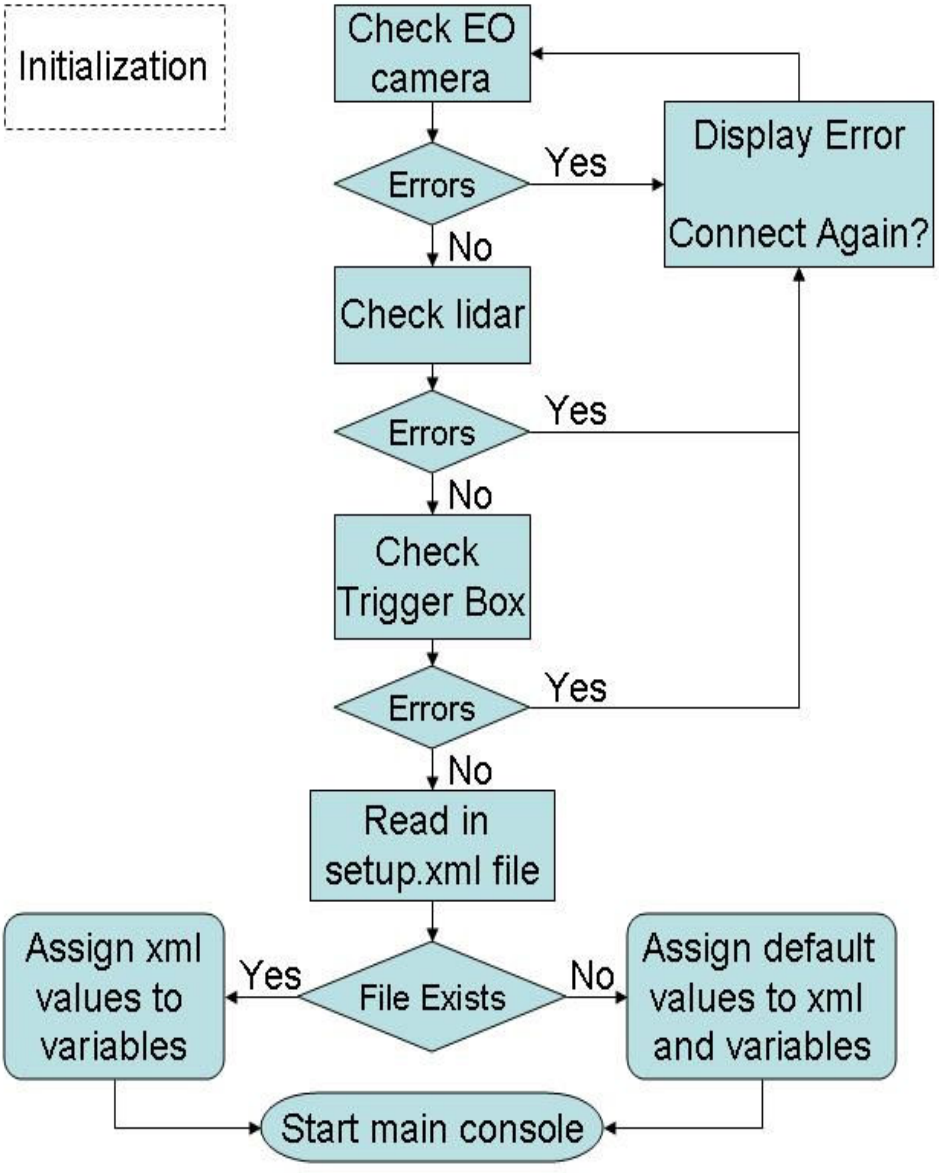

Fig. 4.4. LDScanner™ initialization.

Fig. 4.5. LDScanner™ initialization screen shot.

2) Main Console: When the operator pushes the Setup Scan button on the Main Console, the sub-screen displaying the ready status closes and is replaced with the scan option sub-screen. The scan option sub-screen contains three buttons that allow the operator to choose Read Scan, New Scan or Duplicate Scan, see fig. 4.6.

Clicking on New Scan brings up the Windows OS Save As… dialog box with either the default save name or the name of the previous scan incremented by one. The operator can either use the suggested name or type in a new name as desired.

Clicking the Open… button brings up the Windows Open… dialog box where the operator can select a previous scan file. It is important to note, however, that when using

this option if the files from the previous scan have not been saved to a different folder, they will be overwritten without prompting.

If the Duplicate Scan button is pressed the Windows Open dialog box appears prompting the operator to select the desired scan file. Once selected the Windows Save As… dialog box immediately appears suggesting the name of the selected file incremented by one larger than the largest incremented file. The operator can either accept the suggested name or save the file under a different name. The duplicate scan option allows the operator to run the same scan repeatedly without writing over the previous scan. This is helpful if lighting conditions change or in areas with high traffic to ensure the best image is saved.

3) Scan Console: Any of the three scanning options will close the Main console then open up the Scan Console as shown in fig. 4.7. The change from the Main console to the Scan console appears as a screen refresh to the operator and will probably go unnoticed. The Scan console flow is shown in fig. 4.8.

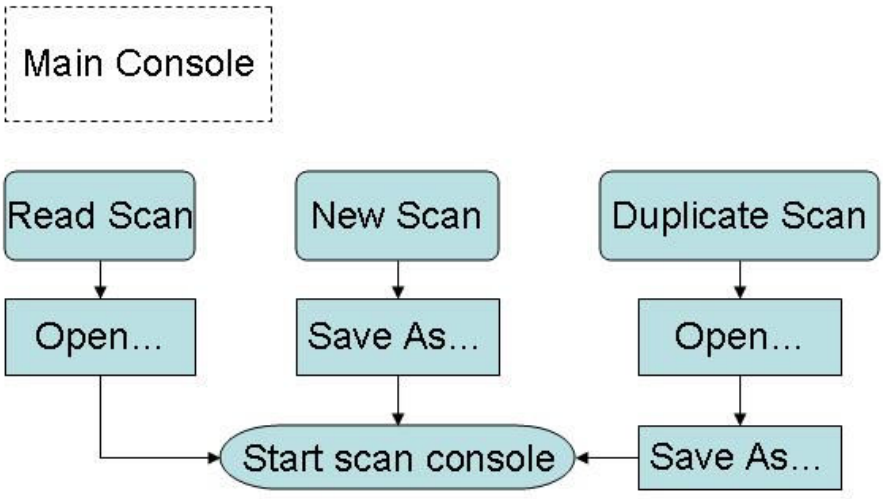

Fig. 4.6. LDScanner™ main console.

Fig. 4.7. LDScanner™ scan console screen shot.

The first sub-screen in the Scan Console prompts the operator for lidar scan parameters. The New Setup scan parameters are name, shot spacing (milliradians), lower and upper azimuth angles (0-360 degrees) and lower and upper elevation angles (50-130 degrees). These parameters can all be typed in directly on this screen.

Alternately, the azimuth angles can be entered by clicking the Pick Limits button located in the center of the bottom portion of the Scan Console as shown in fig. 4.7. By clicking this button the Camera Image dialog box appears and the azimuth points can be selected interactively while viewing the camera output. The camera properties can also be adjusted using this dialog box. The Camera Image dialog box is discussed further in Chapter 5, section D.

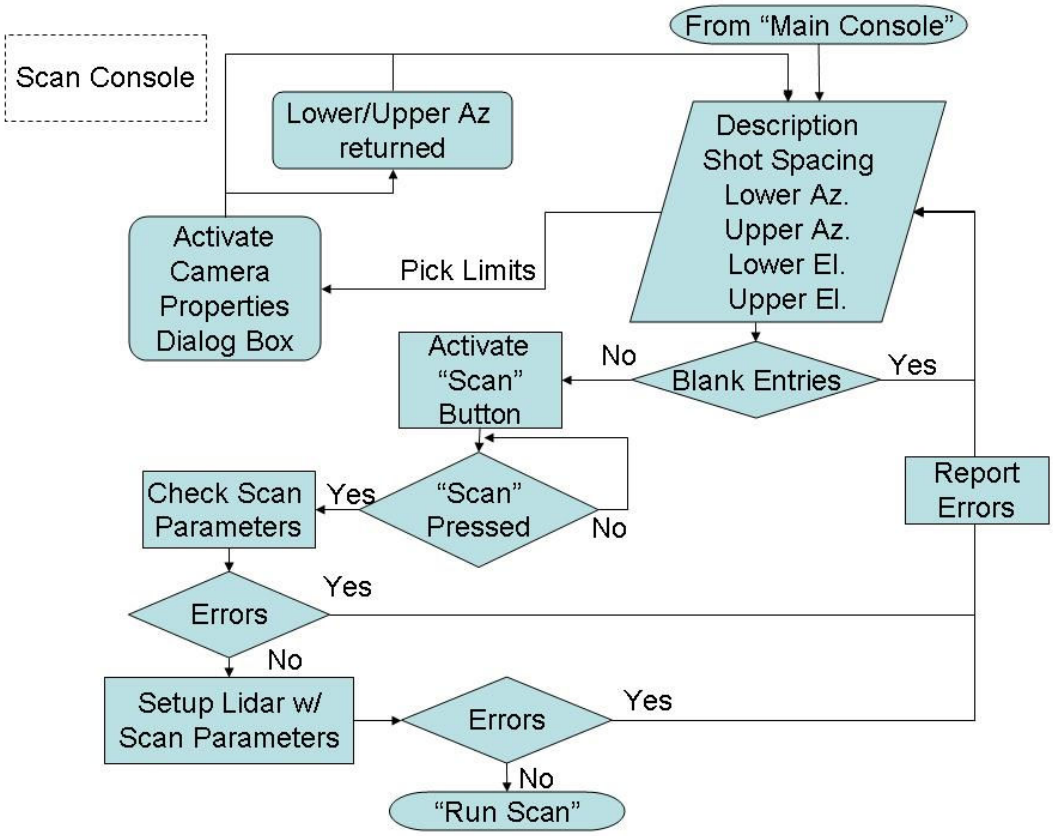

Fig. 4.8. LDScanner™ scan console.

As shown in fig. 4.8, once all the parameters are entered the Scan button becomes active. To conduct a scan the operator presses the Scan button. The Scan Console takes the parameters entered by the operator and checks that all the inputs are valid. If the parameters are valid then they are sent to the lidar. During this process, LDScanner™ monitors replies from the lidar for any problems and notifies the operator as appropriate. If the lidar was programmed successfully, the program moves into the Run Scan block.

4) *Run Scan:* As the Run Scan block starts (see fig. 4.9) a dialog box appears indicating that the lidar was programmed successfully, giving an approximate scan time and prompting whether the operator wants to proceed or cancel the scan. If the operator

clicks Cancel, the Scan Console form is restarted with the scan parameters sub-screen again displayed. The operator can modify the scan parameters and try again or close the program.

If the operator clicks OK, the Scan Console buttons are all deactivated and the scan begins. LDScanner™ is not accessible during the scan. This is to prevent problems during the scan process. An override is provided if the operator wants to interrupt the scan process. By pressing and holding the CTRL+C key combination the console will activate and prompt the operator to stop or continue the scan. If the operator chooses to stop the scan then LDScanner™ will close the files and stop the threads. However, no communication can take place with the lidar until the lidar is finished scanning. The only way to stop the lidar during the scan process is to turn the lidar power off, an action that is not recommended.

While scanning, LDScanner™ stores all collected data into the appropriate lidar (lda) or EO (tex) files and monitors for when the lidar is finished scanning and has returned to the starting position. Once the lidar is finished scanning, LDScanner™ ends all threads. This stops the synchronization box and EO camera and closes all the data files. LDScanner™ moves into the Scan Complete block once everything is stopped and closed.

5) Scan Complete: The Scan Complete portion of LDScanner™ is in fig. 4.10. This portion starts by letting the operator know if there were any errors during the scan. The error is simply displayed for the operator's information and does not affect program flow.

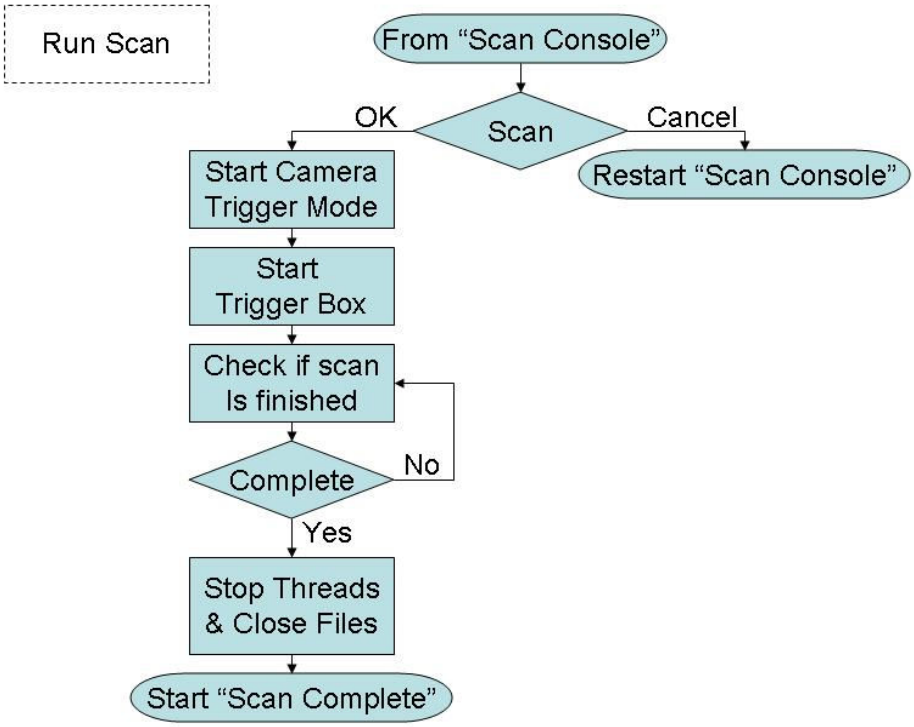

Fig. 4.9. LDScanner™ run scan.

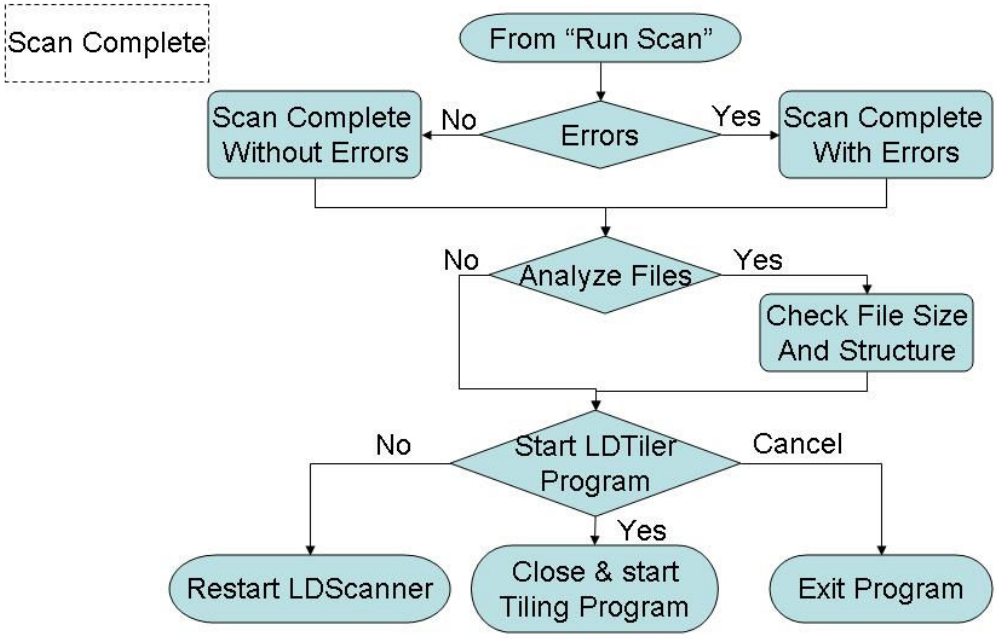

Fig. 4.10. LDScanner™ scan complete.

When the completion of the scan has been acknowledged another dialog box appears that prompts the operator whether or not to analyze the files. The analyzing process checks for correct file format and size. Again, the operator can choose to analyze or not, but the program will continue either way. If the operator chooses to analyze the files, a dialog box will appear when analyzing is finished indicating whether or not the files contain errors.

Finally, the last dialog box that appears asks the operator whether or not to start LDImager™. If the operator clicks Yes, LDScanner™ closes, parks the lidar in the storage position and LDImager™ is started. If the operator clicks No, LDScanner™ simply starts over. If the operator clicks Cancel, LDScanner™ parks the lidar in the storage position and closes.

6) Debugging LDScanner™: While LDScanner™ was being put together and tested; a significant but intermittent bug was encountered. This bug had to do with the threads used in the program. To handle the different simultaneous tasks that were required of the computer a new thread, or concurrent process, was started for each device. As a result there was a thread for the lidar, a thread for the EO camera and a thread for the synchronization box.

There were a number of different manifestations of the thread problem and were all caused by the same error: the threads weren't finishing before the program would move on. A thread timing problem occurred on a thread.suspend and thread.resume check. The program would test whether the thread existed or was paused to determine if it should be started or resumed. If the program determined that the thread was paused,

the resume command was sent but the program would return an error. It was evident that sometime in between the check and sending the command the thread had changed states and so the program would error out.

Another related thread problem occurred when the scan was complete. The program would indicate that the scan was complete before the camera thread could finish. This would result in the EO camera file either closing prematurely or not closing at all.

The solution to these issues was to ensure that the program was able to finish its current task before moving on. These tasks didn't release properly even though they were within a thread which is a protected region. This was believed to be a VB issue. To overcome this problem, a brief delay was added to the program to allow the thread to finish what it was doing before performing the logic test.

7) Advanced Editor: There are several configuration parameters used in LDScanner™. These parameters are stored in a file called setup.xml. When LDScanner™ is started it checks to see if the setup.xml file exists. If the file exists the parameter values stored in the file are assigned to the applicable parameter within the program. If the file does not exist then LDScanner™ will create the file using default values stored within the program.

The setup.xml file contains two tables called CameraSetup and GeneralSetup (see Table 4.1). Within each of the tables are parameters that will change on a regular basis and parameters that are fixed following calibration. The values for Brightness, Analog Gain, ShutterSpeed and NextFileName are variables that are changed within LDScanner™ at runtime. These properties are automatically saved to the setup.xml file

when the operator begins a scan. Every time LDScanner™ starts the setup.xml file is reloaded using the most recent property values. This saves time when multiple scans are being conducted.

The remaining parameters of Height, Width, Bottom, Left, PacketSize, LidarReturn, LidarAmplitudeMax and LidarAmplitudeMin are fixed and are not modified during normal program operation. The Advanced Editor makes these parameters available for editing during calibration. A screen shot of the Advanced editor is shown in fig. 4.11.

Multiple scanning profiles can be created for use in the setup.xml file. The Advanced Editor allows the operator to create, edit and delete these profiles. It also allows the operator to see what the current LDScanner™ settings are and save any of the profiles into the setup.xml file for use as the default settings.

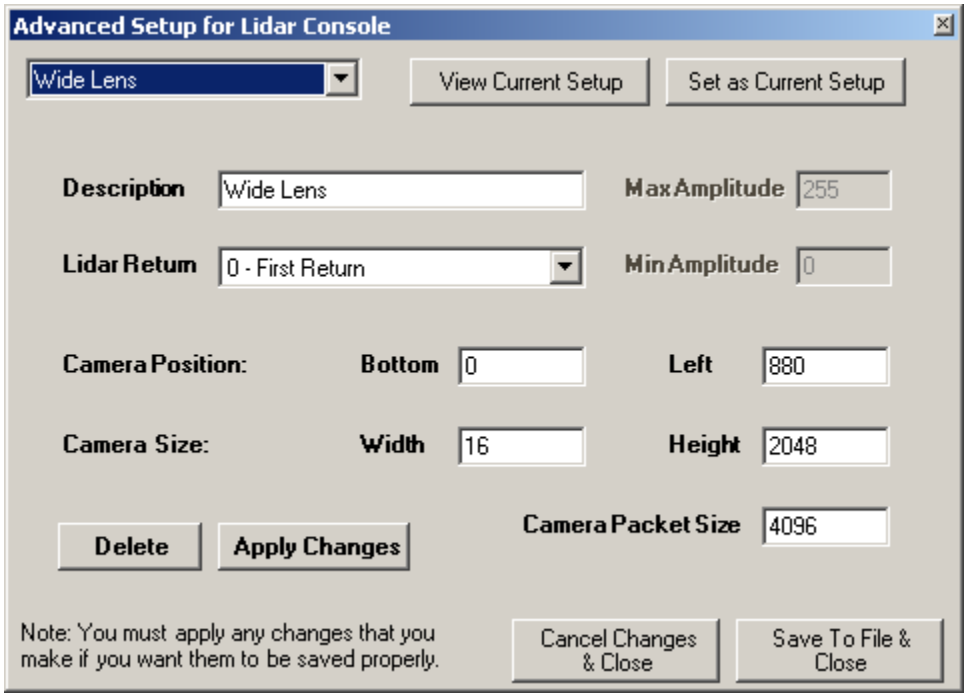

Fig. 4.11. Advanced editor screen shot.

TABLE 4.1
CAMERASETUP AND GENERALSETUP .XML TABLES

CameraSetup Table	
Property	Default Value
Height	2048
Width	16
Bottom	0
Left	880
PacketSize	4096
Brightness	40
AnalogGain	10
ShutterSpeed	40

GeneralSetup Table	
Property	Default Value
NextFileName	lidarscan.lxml
LidarReturn	0
LidarAmplitudeMax	255
LidarAmplitudeMin	0

When the program is started, the Advanced Editor checks to see if consoleprofile.xml and setup.xml exist. If they do not exist then consoleprofile.xml is created with two default profiles called Wide Lens and Narrow Lens. The program then creates setup.xml using the Wide Lens profile which contains the same default values that LDScanner™ uses to create the setup.xml file. If the files exist then the .xml tables are loaded into the editor and the operator can modify the settings from there.

The profile editor allows the operator of the camera to create multiple profiles based on the lens used to capture the images. The lenses are chosen to increase or decrease the effective field of view (FOV) available with the camera. The two default profiles were created for a Wide lens, approximately 80^{o} FOV and a Narrow lens, approximately 40^{o} FOV. The Advanced Editor allows the operator to modify these existing profiles as well as add additional profiles as new lenses are incorporated into the system.

B. LDImager™

LDImager™ converts the raw data sets from the lidar and EO camera, collected by LDScanner™, into a format that can be rendered and viewed in LDModeler™. This is accomplished by creating a 3dd and jpg or other standard lidar and image format files. LDImager™ also allows the operator to analyze any files that have previously been collected to ensure that the file format and size is correct. Once a data set has been converted, LDImager™ allows the user to create a histogram of the jpg image showing the intensity distribution across the entire image. Finally, LDImager™ allows the operator to create, edit and delete profiles used for the conversion. Figure 4.12 shows a screen shot of LDImager™.

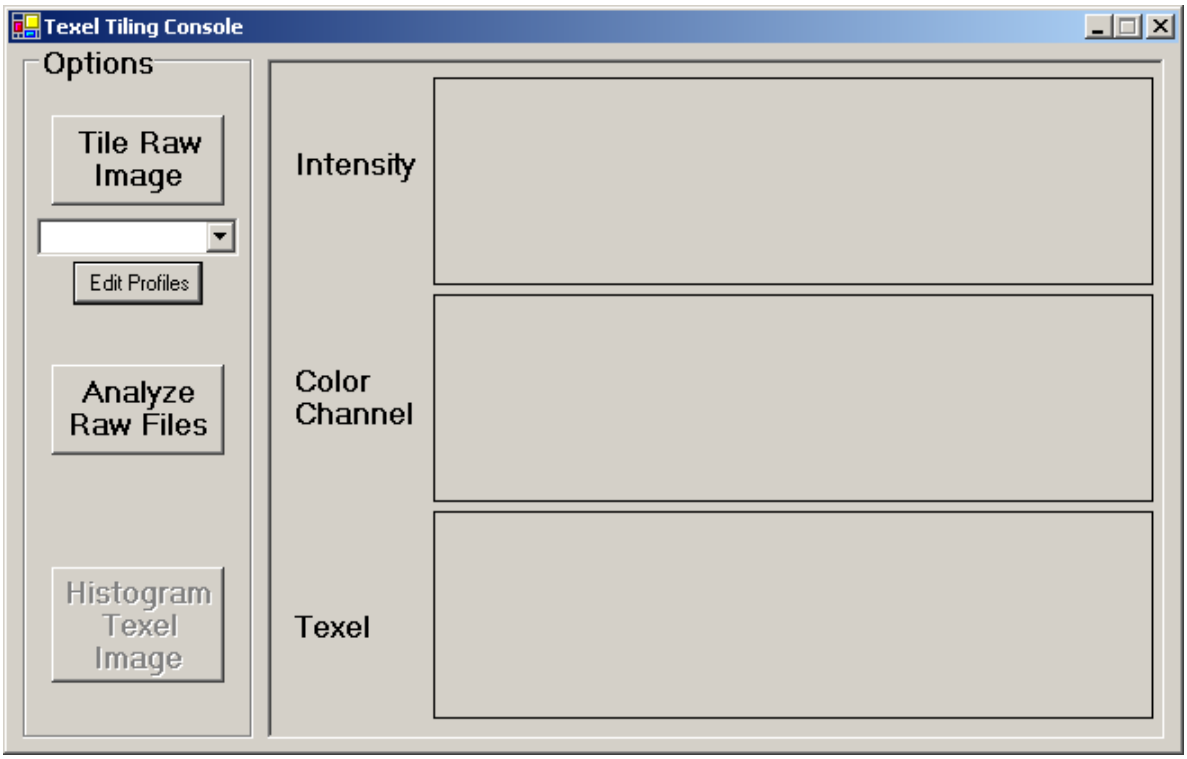

Fig. 4.12. LDImager™ GUI.

LDImager™ has a simple program flow that allows the operator to Tile, Histogram or Analyze the collected EO data. The Histogram option is only available after a data set has been tiled.

Since the Histogram button is initially disabled, the operator really only has two options when the program starts. These options are to Tile or Analyze Raw Files. The program flow is very simple and is shown in fig. 4.13. Each of these steps will be discussed in the following section.

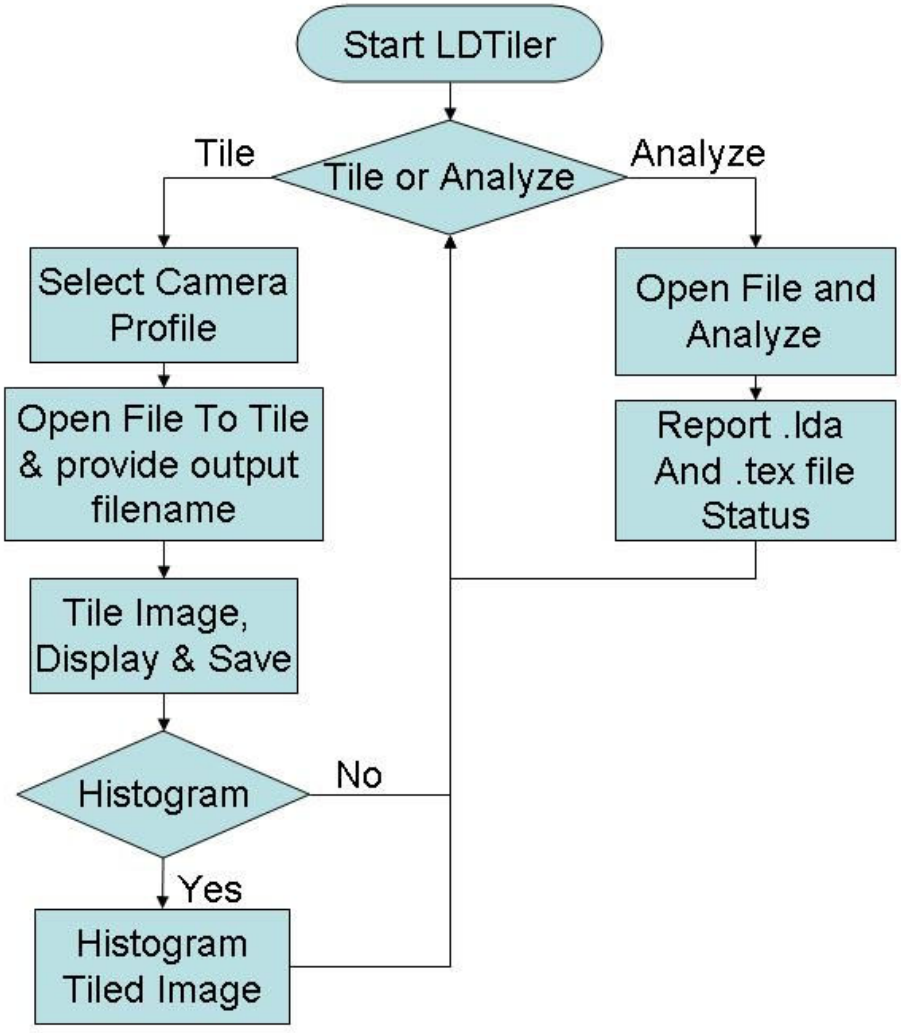

Fig. 4.13. LDImager™ program flow.

1) Tile: If the operator chooses to Tile, the profile must be selected. The profiles are located in the file tileprofile.xml. When LDImager™ first starts it checks for the tileprofile.xml file and if the file exists, the profiles are loaded into the program. If the file does not exist, LDImager™ creates the file with two default profiles called Wide Lens and Narrow Lens and then loads the two profiles into the program. The operator can choose the desired profile by selecting it from the drop down box just below the Tile Raw Image button.

Once the desired profile is selected the operator then selects Tile Raw Image. When this button is pressed the Windows Open dialog box appears. The operator selects the tex file that is to be tiled and then the Windows Save As… dialog box appears. The operator is prompted with a default file name and can choose to accept this file name or use a different name for the lidar and EO output. Once the operator has selected the output file, the tiling process begins by displaying a Tiling… busy box. LDImager™ automatically saves the 3D Texel™ image to file as a jpeg image when the tiling is complete. Then it displays three images to the operator. The intensity image displays the lidar intensity information from the scan. In addition to range and intensity information, the lidar receiver also collects the red, green, and blue (RGB) color information for the individual range points. This color information is displayed in the color channel window. The last window contains the 3D Texel™ image derived from the tiling operation. The operator now has the option to histogram the recently tiled image, tile another image or analyze data sets.

a) Tiling Algorithm: When the data sets are first collected they are not in a

format that is viewable by LDModeler™. The standard formats accepted by LDModeler™ are 3dd and jpg. The lda file from the lidar contains the x-y-z point cloud, color channel and intensity information. The tex file contains hundreds of raw images that are collected at approximately three times the line-scan rate of the lidar data. LDImager™ takes the two data sets and generates the necessary 3dd and jpg files for viewing.

The redundant data in the tex file is necessary because of the lack of synchronization information coming from the lidar. On the previous prototype the lidar had an internal trigger signal that was made available to the camera so that every time the lidar took a picture the camera would take a picture also. Unfortunately, no such trigger signal is available on this scanner. Also, the lidar does not send a signal indicating its status until the scanner has returned to the start position.

To counter these problems, the lidar and EO camera were synchronized through the Synchronization Box by starting both sensors at the same time and generating a known triggering frequency. The lidar frequency is approximately 20 columns per second and the EO camera approximately 60 columns, or frames, per second. The triggering rate of both devices is known to double precision.

The algorithm used for tiling the raw images, developed by Dr. Robert Pack, was coded by Stan Colby of RappidMapper into a dynamically linked library called TexelEng.dll. The entry point into TexelEng.dll and the entire tiling algorithm is through a single function called TexelImageProcessing. This function requires thirteen inputs. These inputs and descriptions of each are shown in fig. 4.14 and can be edited in the

profile editor discussed in Chapter 4, section C.

The LDImager™ GUI provides an interface between the operator and the TexelEng.dll. The parameter values required by the TexelImageProcessing function are stored in the tileprofile.xml file for each individual profile. These values can be modified using the profile editor available within LDImager™.

2) Histogram: Once the 3D Texel™ image has been generated the operator can generate an intensity Histogram of the image. The Histogram is generated by sub-sampling every fourth pixel of the jpg 3D Texel™ image. The results are displayed in a new pop-up window once the histogram process has completed.

3) Analyze: If the operator clicks the Analyze Raw Files button the Windows Open… dialog box appears. The operator selects the desired tex file to analyze and then LDImager™ checks the lda file and tex file format and size compatibility. The results are reported to the operator in a dialog box when completed.

```
TEXELHANDLE ht,          //texel handle
LIDARHANDLE hl,          //lidar handle
int uOffset,             //u-coordinate of optical center of the FPA
int vOffset,             //v-coordinate of optical center of the FPA
double lAzimuthAngle,    //radians, Azimuth shift to align camera with
ladar
double lElevationAngle, //radians, Elevation shift to align camera with
ladar,
double uPixelWidth,      //millimeters for pixel width of the camera
double vPixelHeight,     //millimeters for pixel height of the camera
double focalLength,      //millimeters for focal length of camera lens
int xPixelsPerLidar,     //ratio of final image x vs lidar columns like
5:1
int yPixelsPerLidar,     //ratio of final image y vs lidar rows like 5:1
double lidarTimingRatio,//lidar vs texel ratio normally should be 1.
Int texelFrameOffset     //texel frame offset normally should be 0
```

Fig. 4.14. TexelEng.dll inputs.

C. Create, Edit or Delete LDImager™ Profile

LDImager™ tiles images based on the profile chosen by the operator. Two default profiles are created and stored in the tileprofile.xml file the first time the program is started. LDImager™ provides a GUI interface to the tileprofile.xml file that allows profile creation, editing or deletion. To access the Camera Profile Editor for LDImager™ the operator clicks the Edit Profile button located below the profile drop down box on the main screen.

After clicking the Edit Profile button, the Camera Profile Editor starts and the profile editor, similar to fig. 4.15, is displayed. Initially a profile is not selected. The operator can select one of the default profiles or select the New… option from the Description drop down box. After selecting a profile, the operator can choose to edit or delete the profile. The Advanced Settings can be displayed by checking the Show Advanced Settings checkbox. Changes are saved to the tileprofile.xml file.

D. Prototype II Software Summary

Custom software was developed to control the scan process of the 3D Texel™ camera and then generate 3D standard data files. This software was called LDScanner™ and LDImager™.

LDScanner™ controls the capturing and saving of data from the lidar and EO camera. This program ensures proper operation of all hardware and validates the files upon collection. The interface resembles a field computer with a program flow similar to a Wizard for 3D Texel™ camera setup and operation. The program greatly simplifies the scanning process while still providing customizability. An Advanced Editor was created

to maintain the initialization parameters saved as profiles in consoleprofile.xml and setup.xml.

LDImager™ converts the raw data sets collected by LDScanner™ into standard lidar and texel image formats. This is accomplished by creating a 3dd and jpg file format. LDImager™ also analyzes data files to ensure data integrity and once a data set has been tiled, LDImager™ allows the user to Histogram the jpg image for intensity distribution across the entire image. Tiling profiles can also be created, edited and deleted by the operator using the profile editor in LDImager™. This data can be quality controlled in the field and adjustments can be made to ensure quality before leaving the site.

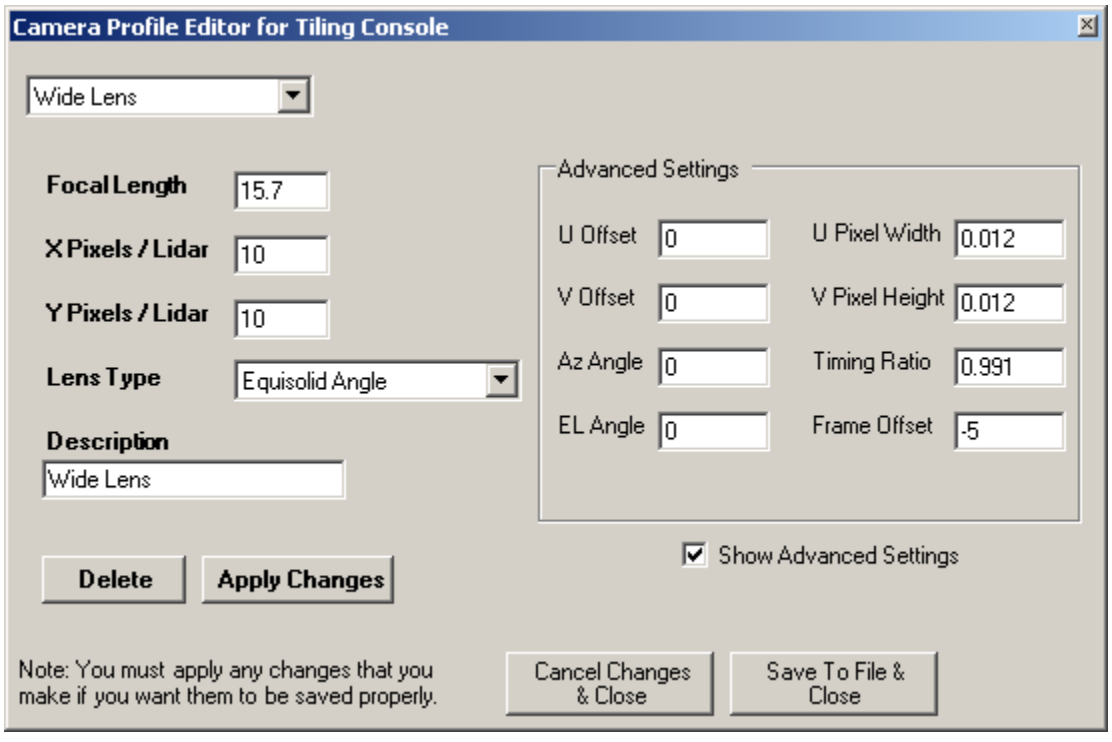

Fig. 4.15. Camera profile editor for LDImager™.

CHAPTER 5

ELECTRO OPTICAL CAMERA INTERFACING

The Prototype II CAIL 3D Texel™ camera uses the ISG LW-3-S-1394-C 'Smart' Digital Imaging Module to collect color imagery. There were several initial approaches to controlling the ISG camera via LDScanner™ but ultimately a modified Carnegie Melon University (CMU) demo program was used [15]. LDScanner™ was written in VB so the modified CMU demo program had to be converted into VB. The CMU1394.dll API has a single entry point through a class declaration and because VB cannot instantiate a class, a wrapper was created that made individual entry points for all of the members and functions within CMU1394.dll. This exposed all members and functions associated with the driver. Once the members and functions were exposed, the camera could be controlled and the Camera Image dialog box was created. The work in this chapter was completed exclusively by the author.

A. Requirements

In Chapter 5 LDScanner™ was introduced as an interface between the operator and the lidar and EO sensors. There is an additional interface between the LDScanner™ software and each of the sensors that handles the low level data flow as shown in fig. 5.1. The focus of the author's exclusive contribution to this project was the construction of the EO camera interface between the LDScanner™ software and the EO camera.

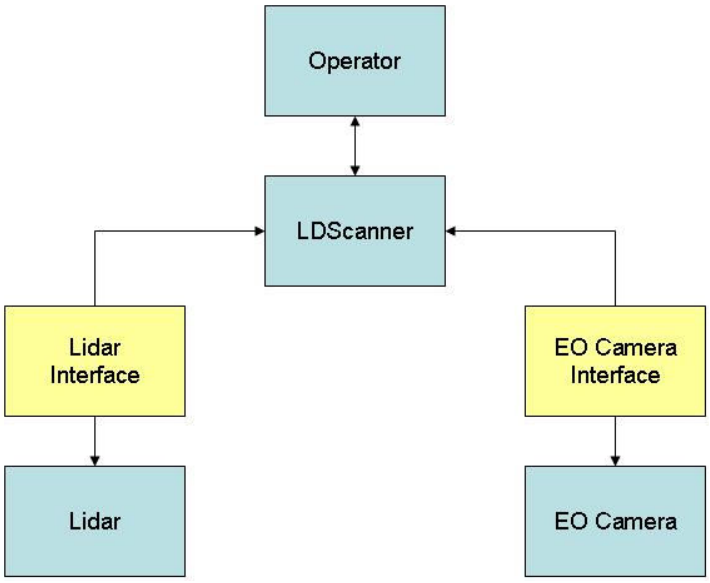

Fig. 5.1. EO camera interface.

The first part of constructing the EO camera interface was to define what LDScanner™ needed to control. The essential requirements are that the EO camera interface must allow control over the following items:

- Region of Interest (ROI), or sub frame
- Position on the CMOS array
- Camera properties
  - Digital gain
  - Brightness
  - White balance
  - Shutter speed
- Triggering
- File format

The ROI is the desired frame size. There are standard frame sizes that can be selected, but LDScanner™ needs to be able to isolate a custom rectangular region, or ROI, and then modify the position of the ROI on the CMOS array. LDScanner™ must also be able to allow the operator to modify the camera properties to adjust the image

quality. In order to coordinate the lidar and EO camera LDScanner™ must also be able to trigger the camera. Finally, the images captured from the camera need to be saved out in a custom file format defined by CAIL.

In addition to these requirements the software needed to be developed as rapidly as possible with a robust implementation. As with any engineering design, there is a trade-off between the development time and robustness requirements. This chapter discusses the solutions that were examined and the implementation of the final solution. The final solution was written in C++ and then made available to VB. The method used to allow VB access to the C++ code and the implementation of the Camera Control dialog box in VB is covered in the latter portion of this chapter.

B. Initial Considerations

When the development of the Prototype II 3D Texel™ camera began the goal was to finish the camera within three weeks. From the beginning it was known that using the CMU1394 API would be the most robust method, but it would require the most time for development. Because of the time limitation, alternative solutions were sought. The alternate solutions examined were Third Party Toolkits, ISG Lightwise demo and CMU demo.

1) Third Party Toolkits: Third party toolkits provide tools that can simplify and expedite the development process. The demo applications of several image toolboxes (IC Imaging Control, Unibrain, ActiveDcam, etc…) were downloaded and installed into Microsoft Visual Studio. The most promising one of these was IC Imaging Control. A sample program was written within a day that allowed a standard 640 (h) x 480 (v) image

to be pulled from the camera in real time at 15 fps and saved to any file format desired. Unfortunately, this toolbox had limited functionality. The ROI could not be modified, the position of the ROI on the CMOS array couldn't be changed, triggering was not available and no camera properties could be modified.

In order to use a third party toolkit, the FireWire™ driver on the camera had to be compatible with the Windows Driver Model (WDM). WDM provides a software interface standard that, when met, controls the communication between the FireWire™ (1394), USB and PCI devices with DirectX/DirectShow. As fig. 5.2 [16] shows, IC Imaging Control, and all third party toolkits examined, interface with the FireWire™ devices through DirectX and thus require the camera driver to be WDM compliant.

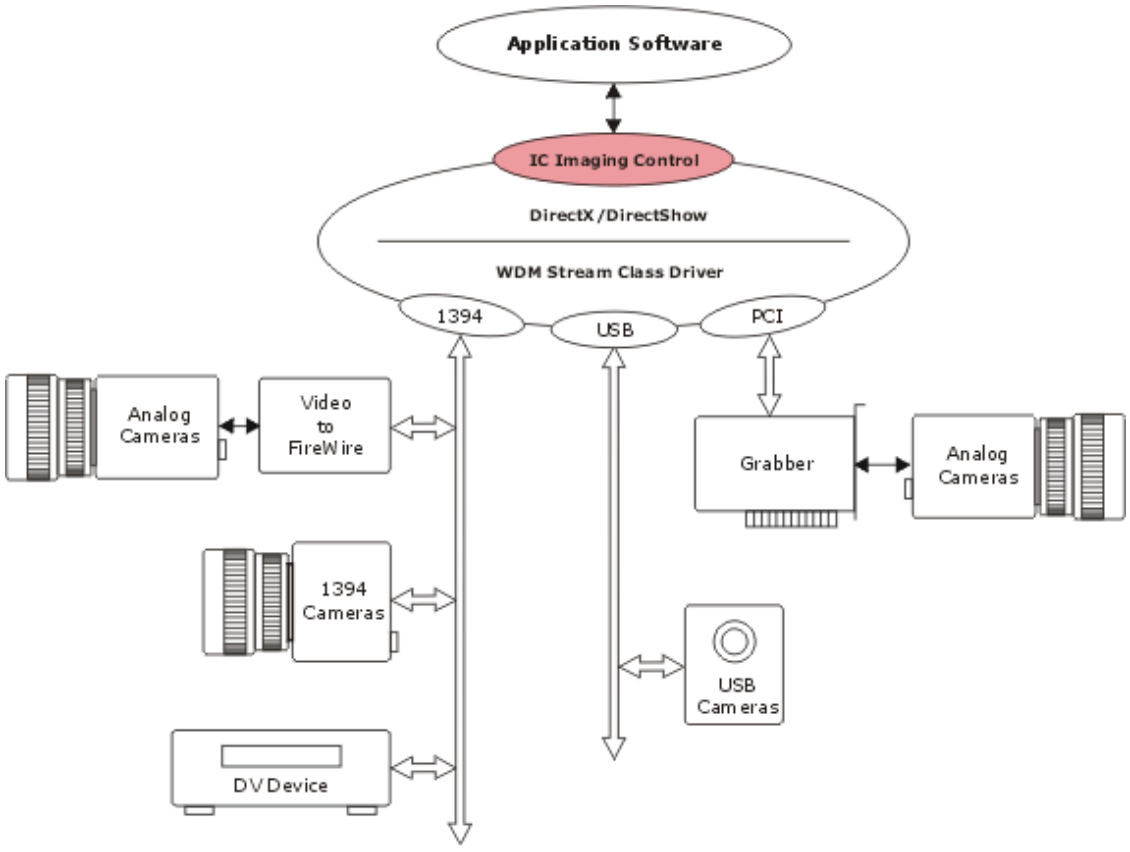

Fig. 5.2. WDM diagram.

The ISG EO camera documentation, however, did not indicate whether it was WDM compliant or not so ISG was contacted [17]. Unfortunately, ISG had not designed their camera around a WDM Stream Class Driver, but instead the ISG camera uses an open source driver from Carnegie Melon University called CMU1394.dll [15]. The driver and API are freely available from The CMU Robotics Institute [18]. By using the CMU driver ISG's camera is not limited to the Windows environment for development, but this also means that none of the third party toolkits could be used.

2) ISG Demo Program: Once it was established that third party imaging software toolkits could not be used, another approach was taken. The ISG camera came with a modified CMU demo program, called Lightwise, which included an additional Camera Control dialog window designed to control the specific features of their camera [19]. The ISG demo program was built for this camera which meant that is should meet all of the camera requirements.

The ISG software includes a Camera Control Dialog box, shown in fig. 5.3, which allows the ROI and its position on the array to be modified dynamically. The software can also control the digital gain, brightness, white balance and shutter speed. There were also several options available for triggering the camera as shown in fig. 5.4. The only requirement that the ISG software didn't meet was to allow for a custom output format, although it was possible to work around this problem.

The biggest disappointment with the ISG software was that, despite it being based on the open source CMU demo, ISG would not release the source code that included the additional Camera Control dialog. This meant that either the program works as-is in the

final solution or it doesn't work at all.

The reason the ISG demo program was not used was that VB could not get a handle on the program nor could the program functions be accessed through the command line. The only way to use the ISG demo program from LDScanner™ was to use the VB SendKeys commands. SendKeys commands are used to emulate the button presses and mouse movement that an operator would use to run a GUI. There is no robust error checking methods within the SendKeys command structure so if anything went wrong, such as the operator pressing a key or minimizing the program while keys are being sent, nothing could be done to remedy the situation. For this reason the ISG demo program was not used.

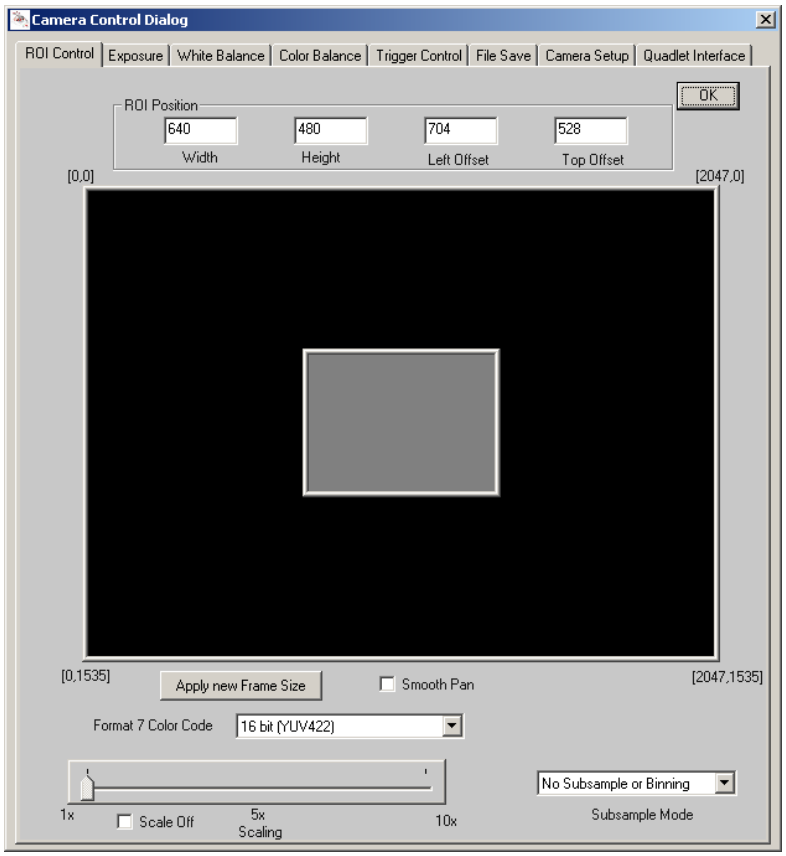

Fig. 5.3. ISG ROI.

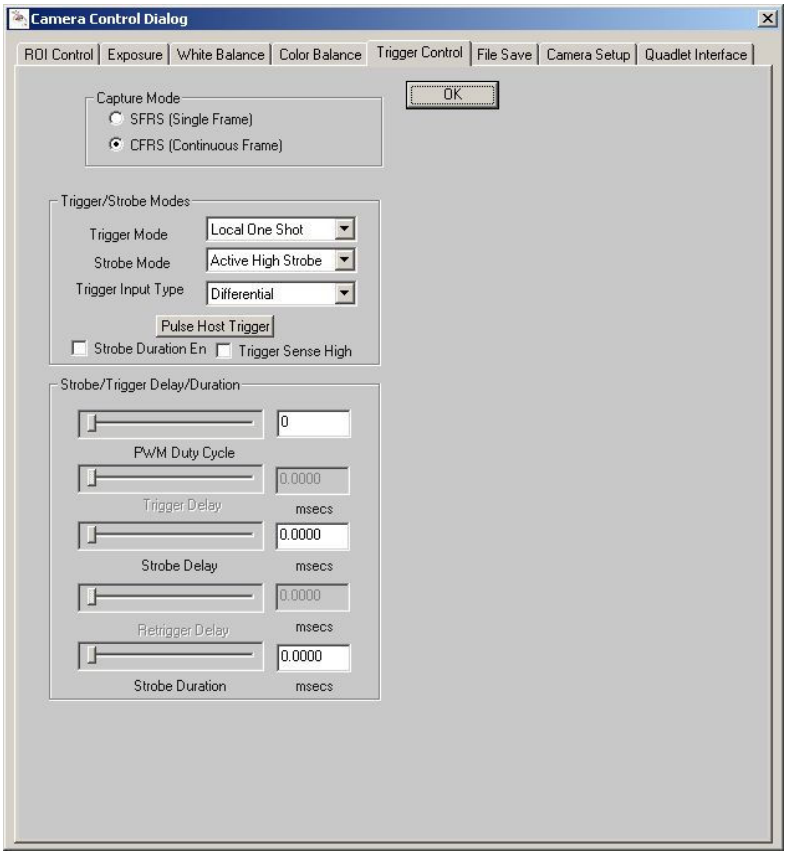

Fig. 5.4. ISG trigger control.

3) CMU Demo Program: The ISG demo program is based on the CMU demo program. Since the ISG program was able to accomplish most of the requirements, it was presumed that the CMU demo program would also be able to perform most of the requirements. Unfortunately, this program suffered the same shortcomings as the ISG demo program did. However, the CMU demo program source code was available and could be edited. The CMU demo program uses Microsoft Foundation Classes (MFC) and a substantial portion of development time was consumed in learning MFC in order to be able to edit the program. Once understood though, the demo program could be easily modified to incorporate all of the requirements. As it turned out, the CMU demo

program could be modified to implement the requirements in a robust manner.

C. Implementation Issues

The modified CMU demo code met all requirements but had problems being incorporated into LDScanner™. The problem was that VB couldn't access the functions from the CMU1394.dll. To overcome this problem a wrapper was created that exposed the functions in CMU1394.dll to VB.

1) Modified CMU Demo: The CMU demo was written so that all of the camera functionality was available to the operator, but the operator had to manually select what the program was to do. To use the demo code in the final solution it needed to simply perform the required tasks when LDScanner™ called it. In other words, the CMU demo program needed to be reduced to a single function. This function needed to Initialize the camera, Acquire Images according to a trigger signal and then Shut Down. The function program flow is shown in fig. 5.5. The code developed in the CMU demo could then be ported to VB and used directly in LDScanner™.

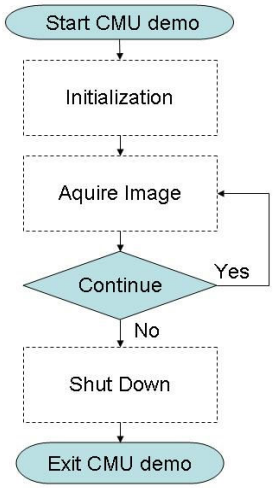

Fig. 5.5. Modified CMU demo program flow.

a) Initialization: Initialization can be broken into two components, the file and the camera initializations. File initialization takes only a few commands and includes the CAIL custom tex file format description. The camera initialization is a little more involved as it covers everything from ensuring that a camera exists to setting camera properties. Most of the code written in the modified CMU demo was associated with initializing the camera.

- *File Initialization:* While the 3D Texel™ camera is operating, the EO camera collects 2048(v) x 16(h) pixels at a frame rate just over 60 fps. A typical scan runs around 20 seconds. That means there can be more than 1200 individual frames for one scan. Having to open and close so many files would slow down the capturing and tiling processes and would be cumbersome to maintain.

Rather than generating hundreds of individual files, a better way of storing all these frames was to put them into one file. This required that a custom file format be defined. The file format took on the extension tex because the file is a collection of texel images, see Appendix E for complete tex documentation. The tex format used in Prototype II has a 16 byte header: 4 bytes for the header size, 4 bytes for the frame width, 4 bytes for the frame height and 4 bytes for the period of the frame rate. This is followed by the individual frames. Each new frame is appended onto the file so that the first frame follows the header and each succeeding frame is added until the function is stopped. The file grows continually as the program is running. When scanning is complete the file is closed and the function ends.

The initialization of this file is straight forward. A file is opened in ab mode

using the fopen command. The ab mode means that all new data will be appended to the end of the file in binary (untranslated) mode so that carriage-return and linefeed characters are suppressed. After opening the file, the header is read and the file is left open. At this point the file initialization is complete and is ready to save images.

- *Camera Initialization:* The camera initialization is broken into general, partial scan (within a designated ROI), and triggering initialization. Regardless of the camera use, the general initialization remains the same. This involves checking to make sure the camera exists and is working properly, selecting the desired camera, initializing the necessary resources and filling in the camera register values. The basic code is shown in fig. 5.6. The critical functions, like CheckLink, provide error checking in case of problems. If the function returns CAM_SUCCESS then the program continues. If there are problems with any of the critical functions, the error must be handled by closing the program. The general initialization must be completed before the partial scan or triggering can be setup.

```
if(theCamera->CheckLink() != CAM_SUCCESS)
    return CAM_ERROR;

if(theCamera->InitCamera() != CAM_SUCCESS)
    return CAM_ERROR;


theCamera->InquireControlRegisters();

theCamera->StatusControlRegisters();
```

Fig. 5.6. EO general initialization.

Next, the partial scan format needs to be set up. Partial scan format is a subclass within the CMU1394.dll driver that designates the ROI and its position on the CMOS array. The first step is to query the camera to see if it supports partial scan format. If a partial scan exists, this function allows the inquire and status functions to be called. These functions determine the camera properties and fill in the appropriate partial scan class member variables.

The format and mode of the camera need to be designated next. The CMU driver supports several standard formats as well as a custom partial scan format, see Table 5.1 [15]. Within each format there are different modes that can be selected. The image size and color format is based on the format and mode selected except in partial scan format. The partial scan format allows customization of the size of the region of interest (ROI), image position, packet size and color format. There are no defined modes within the partial scan format so by setting the mode the operator defines the mode rather than querying the driver for predetermined values. Because of this, any mode, zero through seven, can be selected.

Once the format and mode are selected the specific partial scan settings can be entered. This includes setting the ROI, image position on the focal plane array, packet size and color code. Each image transferred between the camera and computer is broken up into packets. The packet size option defines how large these packets are. The color code option defines whether the image will be output in RGB, YUV, Mono or RAW format.

By being able to customize these options the camera can be configured in many

different ways. However, there are some general guidelines to follow. The camera has a limited ROI granularity value of 8. This means that any ROI can be defined as long as both the height and width are divisible by 8. The ROI for the 3D Texel™ Prototype II was defined as 2048 (v) x 16 (h). The camera is mounted on its side so to allow the long dimension to be vertical. With the camera on its side and using the wide angle lens, the camera had an 80 degree elevation field-of–view which matches the lidar elevation field-of-view.

TABLE 5.1
CMU FORMAT AND MODE

Format	Mode	Image Size	Color Format
0	0	160x120	YUV(4:4:4)
	1	320x240	YUV(4:2:2)
	2	640x480	YUV(4:1:1)
	3	640x480	YUV(4:2:2)
	4	640x480	RGB
	5	640x480	Mono (8-bit)
	6	640x480	Mono (16-bit)
1	0	800x600	YUV(4:2:2)
	1	800x600	RGB
	2	800x600	Mono (8-bit)
	3	1024x768	YUV(4:2:2)
	4	1024x768	RGB
	5	1024x768	Mono (8-bit)
	6	800x600	Mono (16-bit)
	7	1024x768	Mono (16-bit)
2	0	1280x960	YUV(4:2:2)
	1	1280x960	RGB
	2	1280x960	Mono (8-bit)
	3	1600x1200	YUV(4:2:2)
	4	1600x1200	RGB
	5	1600x1200	Mono (8-bit)
	6	1280x960	Mono (16-bit)
	7	1600x1200	Mono (16-bit)
7	0-7	Partial Scan	Selectable

Next, the SetPosition function is called. This function positions the ROI on the array. Initially the ROI was positioned in the center of the array horizontally (1536/2 - 8 = 760) until calibration could be completed. The granularity of the SetPosition command was determined to be 1 so the ROI can be calibrated down to one pixel.

The packet size also needs to be defined. The default packet size is 4096 bytes and works if the full array 2048 (h) x 1536 (v) is used. Since a custom ROI is being used, the packet size needs to be customized. To determine the packet size the equation below was used. The only other requirement is that the packet size does not exceed the default size of 4096 bytes because this is the maximum size that the FireWire™ protocol allows.

$$\mathrm{mod}\left(\frac{Width * Height}{packetsize}\right) = 0.$$

The last item of the partial scan initialization to set up is the color code. The color code options are Raw8, Raw10, RGB, YUV and monochrome. The RGB format was selected for the Prototype II 3D Texel™ camera. The complete partial scan initialization code is shown in fig. 5.7.

```
theCamera->m_controlSize.Supported();
theCamera->m_controlSize.Inquire();
theCamera->m_controlSize.Status();

theCamera->SetVideoFormat(7);
theCamera->SetVideoMode(0);

if(theCamera->m_controlSize.SetSize(width, height) != CAM_SUCCESS)
return CAM_ERROR;
if(theCamera->m_controlSize.SetPosition(left, top) != CAM_SUCCESS)
return CAM_ERROR;
if(theCamera->m_controlSize.SetBytesPerPacket(packetSize) !=
CAM_SUCCESS)
return CAM_ERROR;
if(theCamera->m_controlSize.SetColorCode(4) != CAM_SUCCESS)
return CAM_ERROR;
```

Fig. 5.7. Partial scan initialization.

The last part of the camera initialization is the triggering shown in fig. 5.8. Again the initialize and status functions for the triggering class need to be called to fill in the triggering class member variables from the EO camera. A bug was discovered in the CMU driver on these functions. Both functions are supposed to return CAM_SUCCESS if they work correctly and CAM_ERROR if there are problems. Both functions always return CAM_ERROR when called even though the member variables are filled. Therefore, both functions were called but no error checking was used.

After calling these functions the triggering mode is set. The triggering mode determines whether the input signal expected is a one shot or retriggerable signal and whether the rising or falling edge is used to trigger. The triggering mode selected was for a one shot signal on the rising edge [11]. Once the mode is set, triggering is turned on by setting the TurnOn function to true. With the triggering turned on the camera is now configured and ready to start capturing images.

b) *Image Acquisition:* With the camera initialized and triggering turned on, images can now be acquired. The acquisition process, shown in fig. 5.9, is relatively simple. First, the StartImageAcquisition function is called to initialize the resources necessary for acquiring images then starts the camera streaming [15]. This command is called only once. Once the camera is streaming, images can be read from the camera buffer by using the AcquireImage function. The AcquireImage function grabs a single frame from the camera and places the frame into a buffer called m_pData. The frame is pulled out of m_pData by the getRGB function and then written to file using the fwrite function. The AcquireImage, getRGB and fwrite functions are nested inside a while loop

that runs continually until the scan is complete.

The AcquireImage function won't read until the image buffer on the camera has received a new image. If the AcquireImage function waits for more than ten seconds a timeout occurs and the function returns an error. The coordination of the camera and trigger signal became critical because of the timeout potential. If the AcquireImage function is called and the trigger TurnOn value set to true but the camera is not triggered within ten seconds then the program exits and no images are captured.

c) Shut Down: Once the Acquire Image portion is finished the program begins the shut down phase, see fig. 5.10. This is a simple process that ensures that the camera is shut down properly and the file is closed. To close down the camera the StopImageAcquisition function is called. This function stops streaming video and frees the resources allocated by StartImageAcquisition [15]. No error checking was used on this function because it will return CAM_SUCCESS regardless of whether it has successfully stopped the camera or not. If it does encounter an error in the process, it traces it, but then continues on to free whatever remaining resource(s) it can.

Once the video has been stopped, the trigger TurnOn function is set to false and the file is closed using fclose. The m_pBitmap buffer is then deleted to free up resources and finally, the program exits with CAM_SUCCESS.

```
theCamera->m_controlTrigger.Inquire();
theCamera->m_controlTrigger.Status();
if(theCamera->m_controlTrigger.SetMode(0,0) != CAM_SUCCESS)
return CAM_ERROR;
if(theCamera->m_controlTrigger.TurnOn(true) != CAM_SUCCESS)
return CAM_ERROR;
```

Fig. 5.8. Triggering initialization.

```
while(bView==true)
{
if (theCamera->AcquireImage())
return CAM_ERROR_FRAME_TIMEOUT;

theCamera->getRGB(m_pBitmap);
fwrite(m_pBitmap,1,width * height * 3, fp);

}
```

Fig. 5.9. Acquire image.

```
theCamera->StopImageAcquisition()
theCamera->m_controlTrigger.TurnOn(false);
fclose(fp);
delete [] m pBitmap;
return CAM_SUCCESS;
```

Fig. 5.10. Shut down.

The CMU demo program was thus modified to initialize, acquire images and shut down. Once the CMU API was understood, and this sample program written, it needed to be made available to LDScanner™.

2) Accessing CMU1394.dll From VB: The EO camera was tested and ran successfully using the modified CMU demo program. This program was written in Visual C++. The functions within the CMU 1394 software library work in C++, but LDScanner™ is written in Visual Basic. In order for the final solution to be robust, the software used to control the EO camera had to be integrated into LDScanner™.

Visual Basic allows use of functions within dll's with very little effort as long as each function has its own entry point into the dll. The CMU dll was written so that it could handle multiple cameras. This is possible by keeping all of the functions within a class and declaring a new instance of the class for each new camera. This means that the entry point to all of the CMU functions is through a class. VB cannot instantiate an

instance of a C++ class so the modified CMU demo program could not be converted into VB nor could LDScanner™ get a handle on the program.

a) *Camera.dll Wrapper:* Since Visual Basic has no way of accessing functions contained within the CMU1394.dll driver, a portion of the code would still need to be in C++. Initially it looked like a separate program would have to be developed in C++ and VB would simply execute the program without having a handle on it. After much research, Dan Scofield of the Utah State University Computer Sciences Department presented a solution. The solution was to make a wrapper around the dll that instantiates of the CMU1394.dll class within C++ and then exposes all of the required functions [20].

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include "1394Camera.h"

C1394Camera *theCamera; // class instantiation

int __declspec(dllexport) CALLBACK C1394CameraHeight()
{
return(theCamera->m height);
}

BOOL APIENTRY DllMain( HANDLE hModule,
DWORD  ul_reason_for_call,
LPVOID lpReserved
)
{
switch(ul_reason_for_call)
{
case DLL_PROCESS_ATTACH:
{
theCamera = new C1394Camera();
break;
}
case DLL PROCESS DETACH:
{
delete theCamera;
break;
}
default:
{
break;
}
}
return TRUE;
}
```

Fig. 5.11. CMU wrapper camera.cpp.

Dan Scofield set up the framework for the wrapper shown in fig. 5.11. The wrapper is made up of two files, Camera.cpp and Camera.def. The Camera.cpp file is divided into three main areas. The top portion of the code contains the libraries needed for the dll and a single C1394Camera instantiation called theCamera, the middle portion is an example function declaration and the bottom portion of the code is the entry point into the dll. The Camera.def file is a function definition file. These two files are compiled into a single Camera.dll file.

The basic structure demonstrated how to expose a single member of the CMU1394.dll API within Camera.dll. Using this example the remaining members and functions within CMU1394.dll were exposed by adding function calls in the middle portion of the code. The complete Camera.dll source code is included in Appendix B.

In addition to exposing the individual members and functions from the CMU1394.dll, it was decided that, since the modified CMU demo program was written in C++, the best way to implement this code was to create an additional function within Camera.dll. This function contains the modified CMU demo code and is called StartRecorder, see Appendix A. The StartRecorder function can be started and stopped from VB since it has an individual entry point through Camera.dll. In order to stop the function, a second function was created within Camera.dll called StopRecorder. StopRecorder sets the value bView to false. bView is a Boolean variable used as the test for the while loop within the StartRecorder function. It is a global variable that is initialized to true within StartRecorder to allow the while loop to start.

After all functions were created the wrapper was compiled. This exposed and

made individual entry points available for each member and function in the CMU dll. With the entry points available, this dll could be used by LDScanner™ to control the EO camera.

b) VB Module: To import the newly created functions from the wrapper into VB a module called CMUCamera.vb was created. The format for accessing the Camera.dll function was to declare the function within VB. Just as with any other VB function declaration the input variables and return variable types are defined as well as the functions scope. A sample declaration is shown in fig. 5.12.

One of the biggest problems with importing dll functions into VB is data type integrity. For example, in C++ a bool is 1 byte whereas in VB a bool is 2 bytes. There were a number of different data types used in the CMU driver and it was essential to match by equivalent data type rather than by names. A list of the data types used in C++ and their equivalent in VB is shown in Table 5.2.

Every member variable and function within the CMU dll that needed to be accessed within VB needed to have a function declaration like the one above. The CMUCamera.vb VB module contained all of the dll members and functions as well as the StartRecorder and StopRecorder function declarations. The complete CMUCamera.vb is given in Appendix C. Once this module was constructed the entire CMU1394.dll API could be used in VB just as it is used in C++.

```
Friend Declare Function C1394CameraHeight Lib "Camera.dll" () As Integer
```

Fig. 5.12. C++ function exposure in VB.

TABLE 5.2
DATA TYPES

C++ Data Type	Equivalent VB Data Type	Bytes
Bool	Byte	1
Char	Byte	1
Short	Boolean, Char or Short	2
unsigned short	Short	2
unsigned char*	IntPtr	2
Int	Integer	4
Long	Integer	4
Float	Integer	4
unsigned long	UInt32	4
Double	Double	8
long double	Long	8

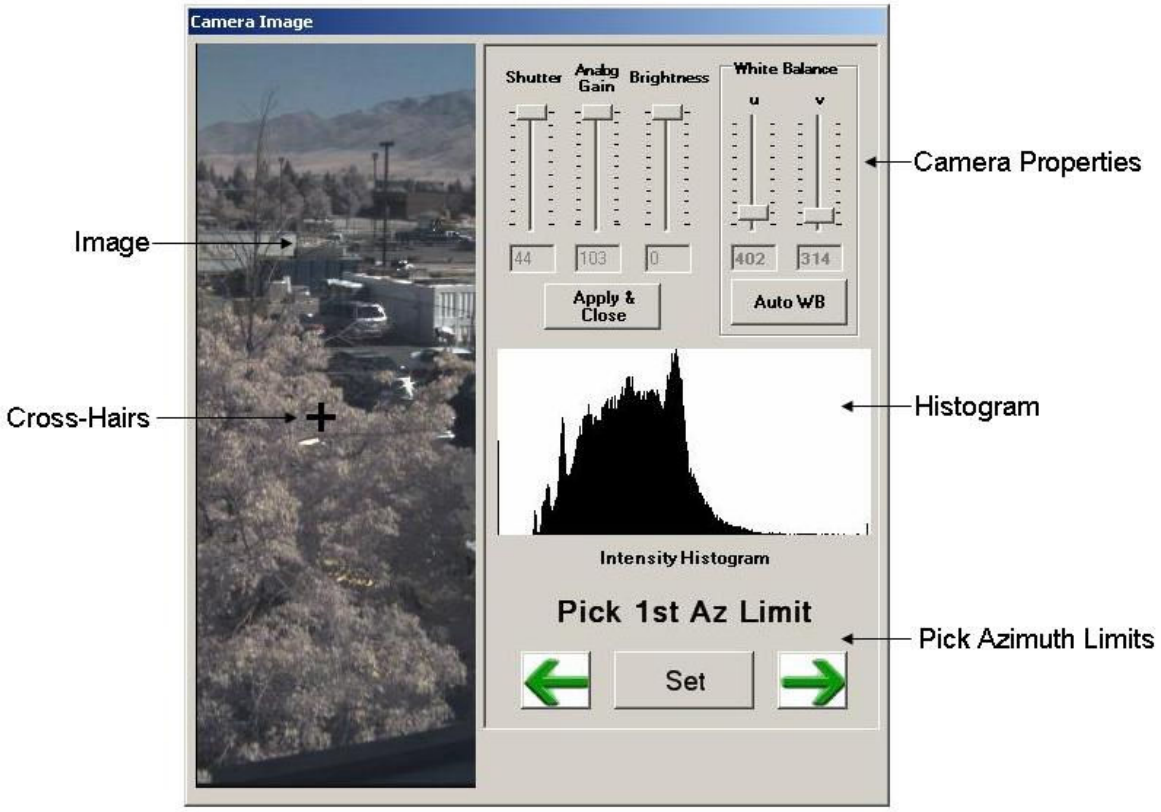

Fig. 5.13. Camera image dialog box.

D. Camera Image Dialog Box

Before the StartRecorder function can be run, the operator needs the opportunity to set the camera properties. The camera properties can be changed by CMU1394.dll through functions that are made available in VB through Camera.dll. To make these functions accessible to the operator a Camera Image dialog box was created. The dialog box contains four areas called Image, Properties, Histogram Plot and Pick Points. These regions are shown in fig. 5.13.

1) Image: The image portion displays a sub-sampled image from the EO camera in real time. The displayed image is similar in shape to the image to be captured, only wider. The width of each frame that will be captured corresponds to the width of the cross-hairs overlaid on the EO image. The cross-hairs also indicate the point that the lidar and EO camera are viewing which is used in the Pick Azimuth Limits portion of the dialog box to set start and end points of scan.

2) Properties: The right hand side of the dialog box contains three items. The portion on the top has the common image property controls of Shutter speed, Analog Gain, Brightness and White Balance. The maximum and minimum range of these controls is determined by querying the camera, except in the case of the shutter speed. The shutter speed had to be limited to within the triggerable region of the camera. A relative, unitless, maximum shutter value of 44 was determined through experimentation.

3) Histogram: The next portion of the dialog box is the intensity histogram region. The intensity histogram is created by converting the RGB data collected from the image into YUV and then displaying the Y, or intensity, value. The equations used to

convert from RGB to YUV are shown below. The intensity information is all that is needed for the histogram so only the Y value was used. YUV values are calculated by

$$Y = (0.299 * R) + (0.587 * G) + (0.114 * B)$$
$$U = 0.493 * (R - Y)$$
$$V = 0.877 * (B - Y).$$

The histogram provides a quantitative representation of the intensity values of the image being displayed. This allows the operator to mathematically determine the quality of the image so that over or under saturation does not occur and thus an optimal image is obtained. The histogram is dynamically scaled so that the chart is relative, not absolute. The ideal histogram image is an even distribution across all intensity values (from 0 to 255 or black to white). Since an even distribution is not normally possible, the EO camera is usually adjusted so values do not pile up on the right side (oversaturated) or on the left side (undersaturated).

4) Pick Limits: The last section of the camera control dialog box contains the pick limits section duplicated from the scan console. These controls are used to setup the scan. The left and right arrows move the lidar clockwise and counter-clockwise when pressed once and continue to rotate until the button is pressed again or the lidar reaches the internal limit switches.

The lidar orientation can be determined by viewing the cross hairs on the image of the Camera Image dialog box or by observing the lidar itself. When the lidar has rotated to the desired position, the operator presses the Set button. When the Set button is pressed the lidar position encoder is queried and the angle, in degrees, is saved. The text

label on the Camera Image dialog box changes from Pick 1st Az Limit to Pick 2nd Az Limit. The operator then rotates the lidar to the second point and presses the Set button again. The lidar position value is saved, the camera dialog box closes and the first and second position values appear in the corresponding blanks on the scan console.

There are two ways to close the Camera Image dialog box. The first is by selecting the azimuth limits for the scan. If, however, the operator wants to change the camera properties without picking azimuth limits, the Apply and Close button will close the dialog box. The complete Camera Image dialog box source code is in Appendix D.

E. EO Camera Summary

The EO camera is used to collect the texture elements for the 3D images. An ISG EO camera that uses the open source CMU1394.dll driver with API was used for the Prototype II 3D Texel™ camera. In order to use this camera, LDScanner™ had to be programmed to control it. There were several initial approaches to controlling the ISG camera but ultimately a modified CMU demo program using the CMU1394.dll API was used. LDScanner™ was written in VB and a wrapper dll was created that made individual entry points for all of the members and functions within CMU1394.dll. The exposed members and functions were then used to control the camera and create the Camera Image dialog box.

CHAPTER 6

RECOMMENDATIONS AND CONCLUSIONS

After the Prototype II 3D Texel™ camera had been designed and constructed it was then calibrated and tested. When the calibration and testing were complete the camera was delivered to RappidMapper for commercial use. A design review was then conducted to discuss the positive and negative aspects of Prototype II. The outcome of this meeting is discussed briefly, followed by some suggestions for future work. Finally, the conclusion summarizes the work presented in this paper.

A. Design Review and Future Work

Prototype II was a huge improvement, in many different ways, over Prototype I. With hardware, the line scan lidar allowed the scan time to be reduced by an order of magnitude while still maintaining high resolution. The FireWire™ Camera simplified the image capturing process and, since an external capture card wasn't needed, also allowed the use of a lightweight laptop computer. This also meant that less power was needed to run the camera so the number, size and consequently the weight of the batteries could be reduced. The cabling, synchronization box, lidar and laptop were designed and built compact and robust which simplified the packing of the hardware components.

The software was simplified and well thought out. This reduced the scan setup time. In addition, a considerable amount of enhancements were made to the program to minimize operator input during instrument operation. Error checking and error handling

were improved and most bugs were removed before release. The design was simple, effective and robust.

Although the design and construction of Prototype II showed much improvement from Prototype I it can still be improved. The biggest hardware concern was the synchronization box. It was designed specifically for this lidar which means that a new synchronization box would need to be designed for any new prototype 3D Texel™ camera built. For this reason it is recommended that a more generalized synchronization box be designed that can be used on Prototype I, Prototype II and potentially any other prototypes.

While redesigning the synchronization box, size should also be a consideration. Functionality more than size was the concern during construction of the first Prototype II camera, but if more of these cameras are to be built the synchronization box would need to be more compact. While reducing the size of the synchronization circuitry, increased integration of that circuitry into the laptop strain relief or the EO camera mount should also be considered.

Another hardware concern was the battery box and charger. Although the design used for the construction of the first Prototype II was adequate, a more robust solution is desirable. For future work, it is recommended that a commercial lithium battery and charger be purchased from an outside manufacturer.

The last hardware suggestion for improvement has to do with the 3D Texel™ camera packaging. Currently two separate boxes, one that stores the lidar and the other that stores everything else but the tripod and gps equipment, are used for the 3D Texel™

camera. A goal during redesign is to reduce the size of as much of the hardware as possible so that only one box is needed for storage and transport.

The software is continually modified and updated based on the current needs, but there is one major improvement that could be made if the LDScanner™ and LDImager™ programs were ever revisited. This improvement would be in the languages chosen to write the software. Currently both of these programs were written in Visual Basic and then integrated with C++ dll's. A better solution would be to rewrite both LDScanner™ and LDImager™ into C++ so that the entire package is in one language. This would remove the need for the language translation modules, Camera.dll and CMUCamera.vb.

B. Conclusion

CAIL has designed and constructed a second prototype integrated lidar/EO camera, or 3D Texel™ camera. This 3D Texel™ camera produces 3D images by synchronizing and aligning a line-scan lidar and EO camera outputs. This paper presented the background of CAIL 3D Texel™ camera technology and the overall design of Prototype II compared with previous work. The specific hardware, software and EO camera interfacing contributed to by the author was explained. The paper concluded with a discussion of the Prototype II design and suggestions for future work on this design.

REFERENCES

[1] R. T. Pack, “A co-boresighted synchronized ladar/EO imager for creating 3D images of dynamic scenes.” The International Society for Optical Engineering, Vol. 5791, Apr. 2005.

[2] Answers.com, Abbreviationz, “LIDAR.” [http://www.answers.com/topic/lidar-1], July 2005.

[3] RIEGL USA, [http://www.riegl.com/terrestrial_scanners/lpm-2k_/lpm_2k_all.htm], Oct. 2005.

[4] PS 12 Volt Batteries, Power Sonic, “Rechargeable Batteries.” [http://www.power-sonic.com], July 2005.

[5] K. Sealy, Personal Interview. Student, Utah State University Center for Advanced Imaging Lidar, July 28, 2005.

[6] Prosilica, “Why Firewire? Camera Link” [http://www.prosilica.com/support/why_firewire.htm], Nov. 2005.

[7] RIEGL Laser Measurement Systems “LMS-Z210i Laser Mirror Scanner: Technical Documentation and Users Instructions.” pg. 29, 2003.

[8] RIEGL USA, [http://www.riegl.com/terrestrial_scanners/lms-z210i_/210i_all.htm], Oct. 2005.

[9] RIEGL Laser Measurement Systems “LMS-Z210i Laser Mirror Scanner: Technical Documentation and Users Instructions.” pg. 14, 2003.

[10] Imaging Solutions Group, “LightWise Camera Series: LW-3-S-1394 FireWire™ Smart Digital Imaging Module.” pg. 3-4, 2004.

[11] Imaging Solutions Group, “LightWise Camera Series: LW-3-S-1394 FireWire™ Smart Digital Imaging Module.” pg. 8-15, 2004.

[12] RIEGL Laser Measurement Systems “LMS-Z210i Laser Mirror Scanner: Technical Documentation and Users Instructions.” pg. 6, 2003.

[13] Powerizer, “14.8 V 2000mah Li-Ion 18650 battery module with Protection IC.” [http://batteryspace.com/index.asp?PageAction=VIEWPROD&ProdID=1328], July 2005.

[14] Hobby Lobby, “Chargers for Lithium Poly Cells/Packs: Apache 1-4 Cell Li-Poly Smart Charger 2500.” [http://www.hobby-lobby.com/chargers_lipoly.htm], July 2005.

[15] C. Baker, “Carnegie Melon 1394 Digital Camera Driver, Help Manual.” Carnegie Melon Robotics Institute, [http://www.cs.cmu.edu/~iwan/1394/1394camera63.zip], Mar. 31, 2004.

[16] IC Imaging Control Group, “IC Imaging Control Technical Concept: How does IC Imaging Control interacts with other components?” [http://www.imagingcontrol.com/ic/docs/faq/how_components_interact.htm], July 2005.

[17] K. Van Isegham, Personal Interview. Founder, Imaging Solutions Group, Feb. 9, 2005.

[18] Carnegie Melon University, Robotics Institute, “IEEE 1394a Driver.” [http://www.cs.cmu.edu/~iwan/1394/], July 2005.

[19] Imaging Solutions Group, “LightWise Camera Series: LW-3-S-1394 FireWire™ Smart Digital Imaging Module.” pg. 33, 2004.

[20] D. Scofield, Personal Interview. Student, Utah State University Computer Science Department, Mar. 7, 2005.

APPENDICES

APPENDIX A

STARTRECORDER FUNCTION

CMU Error Codes [12]

Code	Error	Description
0	CAM_SUCCESS	The operation completed successfully
-1	CAM_ERROR	General Error from the I/O subsystem, try *GetLastError()* to find out what.
1	CAM_ERROR_NOT_INITIALIZED	You forget some required initialization function
2	CAM_ERROR_INVALID_VIDEO_SETTINGS	Somehow, you snuck invalid video settings into the system
3	CAM_ERROR_BUSY	The camera is acquiring or capturing images and whatever you tried to do would break it
4	CAM_ERROR_INSUFFICIENT_RESOURCES	Usually happens when a *GlobalAlloc* fails, which is almost never
5	CAM_ERROR_PARAM_OUT_OF_RANGE	Some parameter you passed is out of range (usually NULL)
6	CAM_ERROR_FRAME_TIMEOUT	The call to AcquireImage() timed out; the current image buffer contains invalid data

```
int __declspec(dllexport) CALLBACK StartRecorder(char* filename, int width, int
height, int left, int top, int packetSize)
{
	// File declarations
	unsigned char *m_pBitmap = new unsigned char[height * width * 3];

	// HEADER: sizeofheader - int(4 bytes), width - int(4 bytes), height
	// - int(4 bytes), time between images in ns - unsigned int (4 bytes)
	BYTE texhead[17];		//Little Endean
	FILE *fp;

	//Header Size
	texhead[0] = 0x10;
	texhead[1] = 0x00;
	texhead[2] = 0x00;
	texhead[3] = 0x00;
	//Width
	texhead[4] = (BYTE)(width & 0x000000FF);
	texhead[5] = (BYTE)((width & 0x0000FF00)>>8);
	texhead[6] = (BYTE)((width & 0x00FF0000)>>16);
	texhead[7] = (BYTE)((width & 0xFF000000)>>24);
	//Height
	texhead[8] = (BYTE)(height & 0x000000FF);
	texhead[9] = (BYTE)((height & 0x0000FF00)>>8);
	texhead[10] = (BYTE)((height & 0x00FF0000)>>16);
	texhead[11] = (BYTE)((height & 0xFF000000)>>24);
	//Time - 60Hz
	texhead[12] = 0;
	texhead[13] = 0x4C;
	texhead[14] = 0xFE;
	texhead[15] = 0;

	//Time - 30Hz
```

```
	//texhead[12] = 0;
	//texhead[13] = 0x98;
	//texhead[14] = 0xFC;
	//texhead[15] = 0x01;

	// Initialize the camera
	if(theCamera->CheckLink() != CAM_SUCCESS)
		return CAM_ERROR;
	if(theCamera->InitCamera() != CAM_SUCCESS)
		return CAM_ERROR;

	theCamera->InquireControlRegisters();
	theCamera->StatusControlRegisters();

	// reads the feature inquiry registers and fills in the
	// corresponding member variables
	theCamera->m_controlSize.Supported();
	theCamera->m_controlSize.Inquire();
	theCamera->m_controlSize.Status();

	// Set partial scan format and mode
	theCamera->SetVideoFormat(7);
	theCamera->SetVideoMode(0);

	if(theCamera->m_controlSize.SetColorCode(4) != CAM_SUCCESS)
	return CAM_ERROR_BUSY;

	// Set partial scan values
	if(theCamera->m_controlSize.SetSize(width, height) != CAM_SUCCESS)
		return CAM_ERROR_NOT_INITIALIZED;
	if(theCamera->m_controlSize.SetPosition(left, top) != CAM_SUCCESS)
		return CAM_ERROR_INVALID_VIDEO_SETTINGS;
	if(theCamera->m_controlSize.SetBytesPerPacket(packetSize) !=
	CAM_SUCCESS)
		return CAM_ERROR_INSUFFICIENT_RESOURCES;

	// Triggering stuff
	//  When trying to error check the Inquire request always returns
	//  CAM_ERROR but the triggering function still works.  So, don't
	//  check for errors on inquire and status
	theCamera->m_controlTrigger.Inquire();
	theCamera->m_controlTrigger.Status();
	if(theCamera->m_controlTrigger.SetMode(0,0) != CAM_SUCCESS)
		return CAM_ERROR_BUSY;
	if(theCamera->m_controlTrigger.TurnOn(true) != CAM_SUCCESS)
		return CAM_ERROR_BUSY;

	// initializes the resources necessary for acquiring images
// and starts the camera streaming. Make sure to use ImageAcquisition
// and NOT ImageCapture
	if (theCamera->StartImageAcquisition())
	{
		theCamera->StopImageAcquisition();
		theCamera->m_controlTrigger.TurnOn(false);
		return CAM_ERROR;
	}

	// Deletes existing file if it exists
	if(remove(filename)==-1);
```

```
	fp = fopen(filename,"ab");
	fwrite(&texhead[0],1,16,fp);

	bView = TRUE;
	while(bView == true)
	{
		if (theCamera->AcquireImage() && bView == true)
		{
			theCamera->StopImageAcquisition();
			theCamera->m_controlTrigger.TurnOn(false);
			fclose(fp);
			delete [] m_pBitmap;
			return CAM_ERROR_FRAME_TIMEOUT;

		}
		else if (bView == true)
		{
			theCamera->getRGB(m_pBitmap);
			fwrite(m_pBitmap,1,width * height * 3, fp);
		}
	}
	// Cleans everything up
	if (theCamera->StopImageAcquisition())
		return CAM_ERROR;
	theCamera->m_controlTrigger.TurnOn(false);
	fclose(fp);
	delete [] m_pBitmap;

	return CAM_SUCCESS;
}
```

APPENDIX B

CAMERA.DLL SOURCE CODE

Camera.def

```
LIBRARY Camera

EXPORTS
      StopRecorder                        @1
      StartRecorder                       @2

      C1394CameraHeight             @3
      C1394CameraWidth              @4
      C1394CamerapData              @5
      C1394CameraLinkChecked        @6
      C1394CameraInitialized        @7

      CheckLinkVB                         @8
      SelectCameraVB                      @9
      InitCameraVB                        @10
      GetVersionVB                        @11
      GetNodeVB                           @12
      GetNumberCamerasVB                  @13
      GetMaxSpeedVB                       @14
      MemGetNumberChannelsVB        @15
      MemGetCurrentChannelVB        @16
      MemLoadChannelVB              @17
      MemSaveChannelVB              @18
      RegLoadSettingsVB             @19
      RegSaveSettingsVB             @20
      ReadQuadletVB                       @21
      WriteQuadletVB                      @22
      GetVideoFormatVB              @23
      SetVideoFormatVB              @24
      GetVideoModeVB                      @25
      SetVideoModeVB                      @26
      GetVideoFrameRateVB                 @27
      SetVideoFrameRateVB                 @28
      StartImageCaptureVB                 @29
      StopImageCaptureVB                  @30
      CaptureImageVB                      @31
      StartImageAcquisitionVB       @32
      StopImageAcquisitionVB        @33
      AcquireImageVB                      @34
      AcquireImageExVB              @35
      getRGBVB                            @36
      getDIBVB                            @37
      YtoRGBVB                            @38
      Y16toRGBVB                          @39
      YUV411toRGBVB                       @40
      YUV422toRGBVB                       @41
```

```
YUV444toRGBVB                       @42
RGB16toRGBVB                        @43
InquireControlRegistersVB   @44
StatusControlRegistersVB    @45
SetBrightnessVB                     @46
SetAutoExposureVB           @47
SetSharpnessVB                      @48
SetWhiteBalanceVB           @49
SetHueVB                            @50
SetSaturationVB                     @51
SetGammaVB                          @52
SetShutterVB                        @53
SetGainVB                           @54
SetIrisVB                           @55
SetFocusVB                          @56
SetZoomVB                           @57

GetBrightnessMin            @58
GetBrightnessMax            @59
GetBrightnessValue1                 @60
GetAutoExposureMin                  @61
GetAutoExposureMax                  @62
GetAutoExposureValue1       @63
GetSharpnessMin                     @64
GetSharpnessMax                     @65
GetSharpnessValue1                  @66
GetWBMin                            @67
GetWBMax                            @68
GetWBValue1                         @69
GetWBValue2                         @70
GetWBOnePushStatus                  @71
GetHueMin                           @72
GetHueMax                           @73
GetHueValue1                        @74
GetSaturationMin            @75
GetSaturationMax            @76
GetSaturationValue1                 @77
GetGammaMin                         @78
GetGammaMax                         @79
GetGammaValue1                      @80
GetShutterMin                       @81
GetShutterMax                       @82
GetShutterValue1            @83
GetGainMin                          @84
GetGainMax                          @85
GetGainValue1                       @86
GetIrisMin                          @87
GetIrisMax                          @88
GetIrisValue1                       @89
GetFocusMin                         @90
GetFocusMax                         @91
GetFocusValue1                      @92
GetZoomMin                          @93
GetZoomMax                          @94
```

```
GetZoomValue1                          @95

SetWBOnePush                           @96

ControlSizemaxV                              @97
ControlSizemaxH                              @98
ControlSizeunitV                       @99
ControlSizeunitH                       @100
ControlSizeunitVpos                          @101
ControlSizeunitHpos                          @102
ControlSizetop                               @103
ControlSizeleft                              @104
ControlSizeheight                      @105
ControlSizewidth                       @106
ControlSizecolorCode                   @107
ControlSizepixelsFrame                 @108
ControlSizebytesFrameHigh              @109
ControlSizebytesFrameLow               @110
ControlSizebytesPacketMin              @111
ControlSizebytesPacketMax              @112
ControlSizebytesPacket                 @113
ControlSizepacketsFrame                @114

SupportedVB                                  @115
ModeSupportedVB                              @116
SetColorCodeVB                               @117
SetSizeVB                                    @118
SetPositionVB                                @119
SetBytesPerPacketVB                          @120
InquireSizeVB                                @121
StatusSizeVB                                 @122

ControlTriggerpresent                  @123
ControlTriggerreadout                  @124
ControlTriggeronoff                          @125
ControlTriggerpolarity                 @126
ControlTriggerMode                           @127
ControlTriggerstatusPolarity  @128
ControlTriggerstatusOnOff              @129
ControlTriggerstatusMode               @130

SetTriggerModeVB                       @131
SetTriggerPolarityVB                   @132
TurnTriggerOnVB                              @133
StatusTriggerVB                              @134
InquireTriggerVB                       @135
```

Camera.cpp

```
#include <windows.h>
#include <stdlib.h>
#include <stdio.h>
#include "1394Camera.h"

// This is the main camera class used to control the
// external device.  It is created here so that the
// DLL can interface with it and give an external
// interface not involving classes.
C1394Camera *theCamera;
bool bView;
extern "C"
/*********************************************************************
**/
/*  StopRecorder  and  StartRecorder  Functions  used  in  ladar  program
*/
/*********************************************************************
**/
void __declspec(dllexport) CALLBACK StopRecorder()
{
      bView = FALSE;
}
int  __declspec(dllexport)  CALLBACK  StartRecorder(char*  filename,  int
width, int height, int left, int top, int packetSize)
{
      // File declarations
      unsigned char *m_pBitmap = new unsigned char[height * width * 3];
      // HEADER: sizeofheader - int(4 bytes), width - int(4 bytes),
      // height - int(4 bytes), time between images in ns -
      // unsigned int (4 bytes)
      BYTE texhead[17];       //Little Endian
      FILE *fp;
      //int number = 0;
      int *lpnDroppedFrames = new int[1];

      //Header Size
      texhead[0] = 0x10;
      texhead[1] = 0x00;
      texhead[2] = 0x00;
      texhead[3] = 0x00;
      //Width
      texhead[4] = (BYTE)(width & 0x000000FF);
      texhead[5] = (BYTE)((width & 0x0000FF00)>>8);
      texhead[6] = (BYTE)((width & 0x00FF0000)>>16);
      texhead[7] = (BYTE)((width & 0xFF000000)>>24);
      //Height
      texhead[8] = (BYTE)(height & 0x000000FF);
      texhead[9] = (BYTE)((height & 0x0000FF00)>>8);
      texhead[10] = (BYTE)((height & 0x00FF0000)>>16);
      texhead[11] = (BYTE)((height & 0xFF000000)>>24);
      //Time - 60Hz
      texhead[12] = 0;
```

```
	texhead[13] = 0x4C;
	texhead[14] = 0xFE;
	texhead[15] = 0;

	//Time - 30Hz
	//texhead[12] = 0;
	//texhead[13] = 0x98;
	//texhead[14] = 0xFC;
	//texhead[15] = 0x01;

	// Initialize the camera
	if(theCamera->CheckLink() != CAM_SUCCESS)
		return CAM_ERROR;
	if(theCamera->InitCamera() != CAM_SUCCESS)
		return CAM_ERROR;

	theCamera->InquireControlRegisters();
	theCamera->StatusControlRegisters();

	// reads the feature inquiry registers and fills in the
	// corresponding member variables
	theCamera->m_controlSize.Supported();
	theCamera->m_controlSize.Inquire();
	theCamera->m_controlSize.Status();

	// Set partial scan format and mode
	theCamera->SetVideoFormat(7);
	theCamera->SetVideoMode(0);

	// RGB color code 4
	if(theCamera->m_controlSize.SetColorCode(4) != CAM_SUCCESS)
		return CAM_ERROR_BUSY;

	// Set partial scan values
	if(theCamera->m_controlSize.SetSize(width,	height)	!=
CAM_SUCCESS)
		return CAM_ERROR_NOT_INITIALIZED;
	if(theCamera->m_controlSize.SetPosition(left,	top)	!=
CAM_SUCCESS)
		return CAM_ERROR_INVALID_VIDEO_SETTINGS;
	if(theCamera->m_controlSize.SetBytesPerPacket(packetSize)
	 != CAM_SUCCESS)
		return CAM_ERROR_INSUFFICIENT_RESOURCES;

	// theCamera->SetVideoFrameRate(5);

	// Triggering stuff
	// When trying to error check the Inquire request always returns
	// CAM_ERROR but the triggering function still works.  So, don't
	// check for errors on inquire and status
	theCamera->m_controlTrigger.Inquire();
	theCamera->m_controlTrigger.Status();

	if(theCamera->m_controlTrigger.SetMode(0,0) != CAM_SUCCESS)
```

```
            return CAM_ERROR_BUSY;
      //if(theCamera->m_controlTrigger.SetPolarity(true)              !=
CAM_SUCCESS)
            //    return CAM_ERROR_INSUFFICIENT_RESOURCES;
      if(theCamera->m_controlTrigger.TurnOn(true) != CAM_SUCCESS)
            return CAM_ERROR_BUSY;

      // initializes the resources necessary for acquiring images and
      // starts the camera streaming
      // Make sure to use ImageAcquisition and NOT ImageCapture
      if (theCamera->StartImageAcquisition())
      {
            theCamera->StopImageAcquisition();
            theCamera->m_controlTrigger.TurnOn(false);
            return CAM_ERROR;
      }

      // Deletes existing file if it exists
      if(remove(filename)==-1);

      fp = fopen(filename,"ab");
      fwrite(&texhead[0],1,16,fp);

      bView = TRUE;
      //size_t NumBytesWritten = 0;
      //size_t buffsize;
      //char outbuff(20);
      //long counter = 0;
      //FILE* outfp;

      //if(remove(strcat(filename,"n"))==-1);
      //outfp = fopen(filename,"a");

      //outfp = fopen(filename,"at");

      // Captures images and saves to file until bView is FALSE
      //  (bView is changed by StopRecorder)
      while(bView == true)
      {
            //if (theCamera->AcquireImage() && bView == true)
            if (theCamera->AcquireImageEx(false, lpnDroppedFrames)
            && bView == true)
            {
                  theCamera->StopImageAcquisition();
                  theCamera->m_controlTrigger.TurnOn(false);
                  fclose(fp);
                  delete [] m_pBitmap;
                  return CAM_ERROR_FRAME_TIMEOUT;

            }
            else if (bView == true)
            {
                  theCamera->getRGB(m_pBitmap);
                  fwrite(m_pBitmap,1,width * height * 3, fp);
```

```
            }
      }

      // Shutdown
      if (theCamera->StopImageAcquisition())
            return CAM_ERROR;
      theCamera->m_controlTrigger.TurnOn(false);
      fclose(fp);
      //fclose(outfp);
      delete [] m_pBitmap;
      return CAM_SUCCESS;
}


/*****************************************************************
**/
/*   Function   definitions   for   the   C1394Camera   Class   Members
*/
/*****************************************************************
**/
int __declspec(dllexport) CALLBACK C1394CameraHeight()
{
      return(theCamera->m_height);
}
int __declspec(dllexport) CALLBACK C1394CameraWidth()
{
      return(theCamera->m_width);
}
unsigned char* __declspec(dllexport) CALLBACK C1394CamerapData()
{
      return(theCamera->m_pData);
}
bool __declspec(dllexport) CALLBACK C1394CameraLinkChecked()
{
      return(theCamera->m_linkChecked);
}
bool __declspec(dllexport) CALLBACK C1394CameraInitialized()
{
      return(theCamera->m_cameraInitialized);
}
/*****************************************************************
**/
/*   Function   definitions   for   the   C1394Camera   Class   Functions
*/
/*****************************************************************
**/
int __declspec(dllexport) CALLBACK CheckLinkVB()
{
      return(theCamera->CheckLink());
}
int __declspec(dllexport) CALLBACK SelectCameraVB(int node)
{
      return(theCamera->SelectCamera(node));
}
```

```
int __declspec(dllexport) CALLBACK InitCameraVB()
{
      return(theCamera->InitCamera());
}
unsigned long __declspec(dllexport) CALLBACK GetVersionVB()
{
      return(theCamera->GetVersion());
}
int __declspec(dllexport) CALLBACK GetNodeVB()
{
      return(theCamera->GetNode());
}
int __declspec(dllexport) CALLBACK GetNumberCamerasVB()
{
      return(theCamera->GetNumberCameras());
}
int __declspec(dllexport) CALLBACK GetMaxSpeedVB()
{
      return(theCamera->GetMaxSpeed());
}
int __declspec(dllexport) CALLBACK MemGetNumberChannelsVB()
{
      return(theCamera->MemGetNumChannels());
}
int __declspec(dllexport) CALLBACK MemGetCurrentChannelVB()
{
      return(theCamera->MemGetCurrentChannel());
}
int __declspec(dllexport) CALLBACK MemLoadChannelVB(int channel)
{
      return(theCamera->MemLoadChannel(channel));
}
int __declspec(dllexport) CALLBACK MemSaveChannelVB(int channel)
{
      return(theCamera->MemSaveChannel(channel));
}
int __declspec(dllexport) CALLBACK RegLoadSettingsVB(const char *pname)
{
      return(theCamera->RegLoadSettings(pname));
}
int __declspec(dllexport) CALLBACK RegSaveSettingsVB(const char *pname)
{
      return(theCamera->RegSaveSettings(pname));
}
int __declspec(dllexport) CALLBACK ReadQuadletVB(unsigned long address,
unsigned long *pdata)
{
      return(theCamera->ReadQuadlet(address, pdata));
}
int   __declspec(dllexport)   CALLBACK   WriteQuadletVB(unsigned   long
address, unsigned long data)
{
      return(theCamera->WriteQuadlet(address,data));
}
```

```
int __declspec(dllexport) CALLBACK GetVideoFormatVB()
{
      return(theCamera->GetVideoFormat());
}
int   __declspec(dllexport)   CALLBACK   SetVideoFormatVB(unsigned   long
format)
{
      return(theCamera->SetVideoFormat(format));
}
int __declspec(dllexport) CALLBACK GetVideoModeVB()
{
      return(theCamera->GetVideoMode());
}
int __declspec(dllexport) CALLBACK SetVideoModeVB(unsigned long mode)
{
      return(theCamera->SetVideoMode(mode));
}
int __declspec(dllexport) CALLBACK GetVideoFrameRateVB()
{
      return(theCamera->GetVideoFrameRate());
}
int __declspec(dllexport) CALLBACK SetVideoFrameRateVB(
      unsigned long framerate)
{
      return(theCamera->SetVideoFrameRate(framerate));
}
int __declspec(dllexport) CALLBACK StartImageCaptureVB()
{
      return(theCamera->StartImageCapture());
}
int __declspec(dllexport) CALLBACK StopImageCaptureVB()
{
      return(theCamera->StopImageCapture());
}
int __declspec(dllexport) CALLBACK CaptureImageVB()
{
      return(theCamera->CaptureImage());
}
int __declspec(dllexport) CALLBACK StartImageAcquisitionVB()
{
      return(theCamera->StartImageAcquisition());
}
int __declspec(dllexport) CALLBACK StopImageAcquisitionVB()
{
      return(theCamera->StopImageAcquisition());
}
int __declspec(dllexport) CALLBACK AcquireImageVB()
{
      return(theCamera->AcquireImage());
}
int __declspec(dllexport) CALLBACK AcquireImageExVB(
      bool DropStaleFrames,
      int *lpnDroppedFrames)
{
```

```
      return(theCamera->AcquireImageEx(DropStaleFrames,
lpnDroppedFrames));
}
void __declspec(dllexport) CALLBACK getRGBVB(unsigned char *pBitmap)
{
      return(theCamera->getRGB(pBitmap));
}
void __declspec(dllexport) CALLBACK getDIBVB(unsigned char *pBitmap)
{
      return(theCamera->getDIB(pBitmap));
}
void __declspec(dllexport) CALLBACK YtoRGBVB(unsigned char *pBitmap)
{
      return(theCamera->YtoRGB(pBitmap));
}
void __declspec(dllexport) CALLBACK Y16toRGBVB(unsigned char *pBitmap)
{
      return(theCamera->Y16toRGB(pBitmap));
}
void    __declspec(dllexport)    CALLBACK    YUV411toRGBVB(unsigned    char
*pBitmap)
{
      return(theCamera->YUV411toRGB(pBitmap));
}
void    __declspec(dllexport)    CALLBACK    YUV422toRGBVB(unsigned    char
*pBitmap)
{
      return(theCamera->YUV422toRGB(pBitmap));
}
void    __declspec(dllexport)    CALLBACK    YUV444toRGBVB(unsigned    char
*pBitmap)
{
      return(theCamera->YUV444toRGB(pBitmap));
}
void    __declspec(dllexport)    CALLBACK    RGB16toRGBVB(unsigned    char
*pBitmap)
{
      return(theCamera->RGB16toRGB(pBitmap));
}
void __declspec(dllexport) CALLBACK InquireControlRegistersVB()
{
      return(theCamera->InquireControlRegisters());
}
void __declspec(dllexport) CALLBACK StatusControlRegistersVB()
{
      return(theCamera->StatusControlRegisters());
}
/* CameraControl wrapper functions */
void __declspec(dllexport) CALLBACK SetBrightnessVB(int value)
{
      return(theCamera->SetBrightness(value));
}
void __declspec(dllexport) CALLBACK SetAutoExposureVB(int value)
{
```

```
      return(theCamera->SetAutoExposure(value));
}
void __declspec(dllexport) CALLBACK SetSharpnessVB(int value)
{
      return(theCamera->SetSharpness(value));
}
void __declspec(dllexport) CALLBACK SetWhiteBalanceVB(int u,int v)
{
      return(theCamera->SetWhiteBalance(u,v));
}
void __declspec(dllexport) CALLBACK SetHueVB(int value)
{
      return(theCamera->SetHue(value));
}
void __declspec(dllexport) CALLBACK SetSaturationVB(int value)
{
      return(theCamera->SetSaturation(value));
}
void __declspec(dllexport) CALLBACK SetGammaVB(int value)
{
      return(theCamera->SetGamma(value));
}
void __declspec(dllexport) CALLBACK SetShutterVB(int value)
{
      return(theCamera->SetShutter(value));
}
void __declspec(dllexport) CALLBACK SetGainVB(int value)
{
      return(theCamera->SetGain(value));
}
void __declspec(dllexport) CALLBACK SetIrisVB(int value)
{
      return(theCamera->SetIris(value));
}
void __declspec(dllexport) CALLBACK SetFocusVB(int value)
{
      return(theCamera->SetFocus(value));
}
void __declspec(dllexport) CALLBACK SetZoomVB(int value)
{
      return(theCamera->SetZoom(value));
}
/*****************************************************************
**/
/*  Function  definitions  for  the  C1394CameraControl  Class  Members
*/
/*****************************************************************
**/
unsigned short __declspec(dllexport) CALLBACK GetBrightnessMin()
{
      return(theCamera->m_controlBrightness.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetBrightnessMax()
{
```

```
	return(theCamera->m_controlBrightness.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetBrightnessValue1()
{
	return(theCamera->m_controlBrightness.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetAutoExposureMin()
{
	return(theCamera->m_controlAutoExposure.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetAutoExposureMax()
{
	return(theCamera->m_controlAutoExposure.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetAutoExposureValue1()
{
	return(theCamera->m_controlAutoExposure.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetSharpnessMin()
{
	return(theCamera->m_controlSharpness.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetSharpnessMax()
{
	return(theCamera->m_controlSharpness.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetSharpnessValue1()
{
	return(theCamera->m_controlSharpness.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetWBMin()
{
	return(theCamera->m_controlWhiteBalance.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetWBMax()
{
	return(theCamera->m_controlWhiteBalance.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetWBValue1()
{
	return(theCamera->m_controlWhiteBalance.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetWBValue2()
{
	return(theCamera->m_controlWhiteBalance.m_value2);
}
bool __declspec(dllexport) CALLBACK GetWBOnePushStatus()
{
	return(theCamera->m_controlWhiteBalance.m_statusOnePush);
}
unsigned short __declspec(dllexport) CALLBACK GetHueMin()
{
	return(theCamera->m_controlHue.m_min);
}
```

```
unsigned short __declspec(dllexport) CALLBACK GetHueMax()
{
      return(theCamera->m_controlHue.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetHueValue1()
{
      return(theCamera->m_controlHue.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetSaturationMin()
{
      return(theCamera->m_controlSaturation.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetSaturationMax()
{
      return(theCamera->m_controlSaturation.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetSaturationValue1()
{
      return(theCamera->m_controlSaturation.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetGammaMin()
{
      return(theCamera->m_controlGamma.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetGammaMax()
{
      return(theCamera->m_controlGamma.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetGammaValue1()
{
      return(theCamera->m_controlGamma.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetShutterMin()
{
      return(theCamera->m_controlShutter.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetShutterMax()
{
      return(theCamera->m_controlShutter.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetShutterValue1()
{
      return(theCamera->m_controlShutter.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetGainMin()
{
      return(theCamera->m_controlGain.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetGainMax()
{
      return(theCamera->m_controlGain.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetGainValue1()
{
```

```
      return(theCamera->m_controlGain.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetIrisMin()
{
      return(theCamera->m_controlIris.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetIrisMax()
{
      return(theCamera->m_controlIris.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetIrisValue1()
{
      return(theCamera->m_controlIris.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetFocusMin()
{
      return(theCamera->m_controlFocus.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetFocusMax()
{
      return(theCamera->m_controlFocus.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetFocusValue1()
{
      return(theCamera->m_controlFocus.m_value1);
}
unsigned short __declspec(dllexport) CALLBACK GetZoomMin()
{
      return(theCamera->m_controlZoom.m_min);
}
unsigned short __declspec(dllexport) CALLBACK GetZoomMax()
{
      return(theCamera->m_controlZoom.m_max);
}
unsigned short __declspec(dllexport) CALLBACK GetZoomValue1()
{
      return(theCamera->m_controlZoom.m_value1);
}
/*****************************************************************
**/
/*  Function definitions for the C1394CameraControl Class Functions
*/
/*****************************************************************
**/
int __declspec(dllexport) CALLBACK SetWBOnePush()
{
      return(theCamera->m_controlWhiteBalance.SetOnePush());
}
/*****************************************************************
**/
/*  Function definitions for the C1394CameraControlSize Class Member
*/
/*****************************************************************
**/
```

```
int __declspec(dllexport) CALLBACK ControlSizemaxV()
{
      return(theCamera->m_controlSize.m_maxV);
}
int __declspec(dllexport) CALLBACK ControlSizemaxH()
{
      return(theCamera->m_controlSize.m_maxH);
}
int __declspec(dllexport) CALLBACK ControlSizeunitV()
{
      return(theCamera->m_controlSize.m_unitV);
}
int __declspec(dllexport) CALLBACK ControlSizeunitH()
{
      return(theCamera->m_controlSize.m_unitH);
}
int __declspec(dllexport) CALLBACK ControlSizeunitVpos()
{
      return(theCamera->m_controlSize.m_unitVpos);
}
int __declspec(dllexport) CALLBACK ControlSizeunitHpos()
{
      return(theCamera->m_controlSize.m_unitHpos);
}
int __declspec(dllexport) CALLBACK ControlSizetop()
{
      return(theCamera->m_controlSize.m_top);
}
int __declspec(dllexport) CALLBACK ControlSizeleft()
{
      return(theCamera->m_controlSize.m_left);
}
int __declspec(dllexport) CALLBACK ControlSizeheight()
{
      return(theCamera->m_controlSize.m_height);
}
int __declspec(dllexport) CALLBACK ControlSizewidth()
{
      return(theCamera->m_controlSize.m_width);
}
int __declspec(dllexport) CALLBACK ControlSizecolorCode()
{
      return(theCamera->m_controlSize.m_colorCode);
}
int __declspec(dllexport) CALLBACK ControlSizepixelsFrame()
{
      return(theCamera->m_controlSize.m_pixelsFrame);
}
int __declspec(dllexport) CALLBACK ControlSizebytesFrameHigh()
{
      return(theCamera->m_controlSize.m_bytesFrameHigh);
}
int __declspec(dllexport) CALLBACK ControlSizebytesFrameLow()
```

```
{
      return(theCamera->m_controlSize.m_bytesFrameLow);
}
int __declspec(dllexport) CALLBACK ControlSizebytesPacketMin()
{
      return(theCamera->m_controlSize.m_bytesPacketMin);
}
int __declspec(dllexport) CALLBACK ControlSizebytesPacketMax()
{
      return(theCamera->m_controlSize.m_bytesPacketMax);
}
int __declspec(dllexport) CALLBACK ControlSizebytesPacket()
{
      return(theCamera->m_controlSize.m_bytesPacket);
}
int __declspec(dllexport) CALLBACK ControlSizepacketsFrame()
{
      return(theCamera->m_controlSize.m_packetsFrame);
}
/*****************************************************************
/
/*   Function   definitions   for   the   C1394CameraControlSize   Class
Functions*/
/*****************************************************************
/
bool __declspec(dllexport) CALLBACK SupportedVB()
{
      return(theCamera->m_controlSize.Supported());
}
bool __declspec(dllexport) CALLBACK ModeSupportedVB(int mode)
{
      return(theCamera->m_controlSize.ModeSupported(mode));
}
int __declspec(dllexport) CALLBACK SetColorCodeVB(int code)
{
      return(theCamera->m_controlSize.SetColorCode(code));
}
int __declspec(dllexport) CALLBACK SetSizeVB(int width, int height)
{
      theCamera->m_controlSize.Status();
      theCamera->m_controlSize.Inquire();
      return(theCamera->m_controlSize.SetSize(width, height));
}
int __declspec(dllexport) CALLBACK SetPositionVB(int left, int top)
{
      return(theCamera->m_controlSize.SetPosition(left,top));
}
int __declspec(dllexport) CALLBACK SetBytesPerPacketVB(int bytes)
{
      return(theCamera->m_controlSize.SetBytesPerPacket(bytes));
}
int __declspec(dllexport) CALLBACK InquireSizeVB()
{
      return(theCamera->m_controlSize.Inquire());
```

```
}
int __declspec(dllexport) CALLBACK StatusSizeVB()
{
      return(theCamera->m_controlSize.Status());
}
/*****************************************************************
**/
/* Function definitions for the C1394CameraControlTrigger Class Methods
*/
/*****************************************************************
**/
bool __declspec(dllexport) CALLBACK ControlTriggerpresent()
{
      return(theCamera->m_controlTrigger.m_present);
}
bool __declspec(dllexport) CALLBACK ControlTriggerreadout()
{
      return(theCamera->m_controlTrigger.m_readout);
}
bool __declspec(dllexport) CALLBACK ControlTriggeronoff()
{
      return(theCamera->m_controlTrigger.m_onoff);
}
bool __declspec(dllexport) CALLBACK ControlTriggerpolarity()
{
      return(theCamera->m_controlTrigger.m_polarity);
}
bool* __declspec(dllexport) CALLBACK ControlTriggerMode()
{
      return(theCamera->m_controlTrigger.m_triggerMode);
}
bool __declspec(dllexport) CALLBACK ControlTriggerstatusPolarity()
{
      return(theCamera->m_controlTrigger.m_statusPolarity);
}
bool __declspec(dllexport) CALLBACK ControlTriggerstatusOnOff()
{
      return(theCamera->m_controlTrigger.m_statusOnOff);
}
int __declspec(dllexport) CALLBACK ControlTriggerstatusMode()
{
      return(theCamera->m_controlTrigger.m_statusMode);
}
/*****************************************************************
**/
/*Function definitions for the C1394CameraControlTrigger Class
Functions*/
/*****************************************************************
**/
int __declspec(dllexport) CALLBACK SetTriggerModeVB(int mode, int
parameter)
{
      return(theCamera->m_controlTrigger.SetMode(mode, parameter));
}
```

```
int __declspec(dllexport) CALLBACK SetTriggerPolarityVB(bool polarity)
{
      return(theCamera->m_controlTrigger.SetPolarity(polarity));
}
int __declspec(dllexport) CALLBACK TurnTriggerOnVB(bool on)
{
      return(theCamera->m_controlTrigger.TurnOn(on));
}
int __declspec(dllexport) CALLBACK StatusTriggerVB()
{
      return(theCamera->m_controlTrigger.Status());
}
int __declspec(dllexport) CALLBACK InquireTriggerVB()
{
      return(theCamera->m_controlTrigger.Inquire());
}
/************************************************************
** DllMain
**    This routine is called when the DLL is loaded, unloaded
**    etc.  It is also called when threads are entered and
**    exited (not dealt with here)
************************************************************/
//extern "C"
BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                               )
{
      switch(ul_reason_for_call)
      {
            case DLL_PROCESS_ATTACH:
            {
                  // Code to run when the DLL is loaded
                  // Create camera class here
            //    MessageBox(NULL,   "A   theCamera   class   has   been
instantiated", "Information",
        //MB_OK);
                  theCamera = new C1394Camera();
                  break;
            }
            case DLL_PROCESS_DETACH:
            {
                  // Code to run when the DLL is freed
                  // Destroy camera class here
                  //MessageBox(NULL,  "theCamera  destructor  has  been
called", "Information",
        //MB_OK);
                  delete theCamera;
                  break;
            }
            case DLL_THREAD_ATTACH:
            {
            // Code to run when a thread is created during the DLL's
lifetime.
```

```
            break;
            }
      case DLL_THREAD_DETACH:
            {
            // Code to run when a thread ends normally.
                  break;
            }
            default:
            {
                  break;
            }
      }

      return TRUE;
}
```

APPENDIX C

CMUCAMERA.VB SOURCE CODE

```
Module CMUCamera
#Region "Camera DLL Constants"
    Friend Const CAM_SUCCESS As Int16 = 0
    Friend Const CAM_ERROR As Int16 = -1
    Friend Const CAM_ERROR_NOT_INITIALIZED As Int16 = 1
    Friend Const CAM_ERROR_INVALID_VIDEO_SETTINGS As Int16 = 2
    Friend Const CAM_ERROR_BUSY As Int16 = 3
    Friend Const CAM_ERROR_INSUFFICIENT_RESOURCES As Int16 = 4
    Friend Const CAM_ERROR_PARAM_OUT_OF_RANGE As Int16 = 5
    Friend Const CAM_ERROR_FRAME_TIMEOUT As Int16 = 6
#End Region

    'Recording Functions
    Friend Declare Sub StopRecorder Lib "Camera.dll" ()
    Friend Declare Function StartRecorder Lib "Camera.dll" ( _
      ByVal filename As String, _
      ByVal width As Integer, _
      ByVal height As Integer, _
      ByVal left As Integer, _
      ByVal top As Integer, _
      ByVal packetSize As Integer _
      ) As Integer

    'Function definitions for the C1394Camera Class Members
    Friend Declare Function C1394CameraHeight Lib "Camera.dll" () _
      As Integer
    Friend Declare Function C1394CameraWidth Lib "Camera.dll" () _
      As Integer
    Friend Declare Function C1394CamerapData Lib "Camera.dll" () _
      As Byte 'Pointer Issues
    Friend Declare Function C1394CameraLinkChecked Lib "Camera.dll" ()
_
      As Byte
    Friend Declare Function C1394CameraInitialized Lib "Camera.dll" ()
_
      As Byte

    'Function definitions for the C1394Camera Class Functions
    Friend Declare Function CheckLinkVB Lib "Camera.dll" () As Integer
    Friend Declare Function SelectCameraVB Lib "Camera.dll" ( _
      ByVal node As Integer
      ) As Integer
    Friend Declare Function InitCameraVB Lib "Camera.dll" () As Integer
    Friend Declare Function GetVersionVB Lib "Camera.dll" () As UInt32
    Friend Declare Function GetNodeVB Lib "Camera.dll" () As Integer
    Friend Declare Function GetNumberCamerasVB Lib "Camera.dll" () _
      As Integer
    Friend Declare Function GetMaxSpeedVB Lib "Camera.dll" () As
```

```
Integer
    Friend Declare Function MemGetNumberChannelsVB Lib "Camera.dll" ()
_
      As Integer
    Friend Declare Function MemGetCurrentChannelVB Lib "Camera.dll" ()
_
      As Integer
    Friend Declare Function MemLoadChannelVB Lib "Camera.dll" ( _
      ByVal channel As Integer _
      ) As Integer
    Friend Declare Function MemSaveChannelVB Lib "Camera.dll" ( _
      ByVal channel As Integer _
      ) As Integer
    Friend Declare Function RegLoadSettingsVB Lib "Camera.dll" ( _
      ByRef pname As Byte _
      ) As Integer
    Friend Declare Function RegSaveSettingsVB Lib "Camera.dll" ( _
      ByRef pname As Byte _
      ) As Integer
    Friend Declare Function ReadQuadletVB Lib "Camera.dll" ( _
      ByVal address As UInt32, _
      ByRef pdata As UInt32 _
      ) As Integer
    Friend Declare Function WriteQuadletVB Lib "Camera.dll" ( _
      ByVal address As UInt32, _
      ByVal pdata As UInt32 _
      ) As Integer
    Friend Declare Function GetVideoFormatVB Lib "Camera.dll" ( _
      ) As Integer
    Friend Declare Function SetVideoFormatVB Lib "Camera.dll" ( _
      ByVal format As UInt32 _
      ) As Integer
    Friend Declare Function GetVideoModeVB Lib "Camera.dll" () As
Integer
    Friend Declare Function SetVideoModeVB Lib "Camera.dll" ( _
      ByVal mode As UInt32 _
      ) As Integer
    Friend Declare Function GetVideoFrameRateVB Lib "Camera.dll" ( _
      ) As Integer
    Friend Declare Function SetVideoFrameRateVB Lib "Camera.dll" ( _
      ByVal frame_rate As UInt32 _
      ) As Integer
    Friend Declare Function StartImageCaptureVB Lib "Camera.dll" ( _
      ) As Integer
    Friend Declare Function StopImageCaptureVB Lib "Camera.dll" ( _
      ) As Integer
    Friend Declare Function CaptureImageVB Lib "Camera.dll" () As
Integer
    Friend Declare Function StartImageAcquisitionVB Lib "Camera.dll" (
_
      ) As Integer
    Friend Declare Function StopImageAcquisitionVB Lib "Camera.dll" ( _
      ) As Integer
    Friend Declare Function AcquireImageVB Lib "Camera.dll" () As
```

```
Integer
    Friend Declare Function AcquireImageExVB Lib "Camera.dll" ( _
      ByVal DropStaleFrames As Byte, _
      ByRef lpnDroppedFrames As Integer _
      ) As Integer
    Friend Declare Sub getRGBVB Lib "Camera.dll" (ByVal pBitmap() As
Byte)
    Friend Declare Sub getDIBVB Lib "Camera.dll" (ByVal pBitmap() As
Byte)
    Friend Declare Sub YtoRGBVB Lib "Camera.dll" (ByVal pBitmap() As
Byte)
    Friend Declare Sub Y16toRGBVB Lib "Camera.dll" ( _
      ByVal pBitmap() As Byte)
    Friend Declare Sub YUV411toRGBVB Lib "Camera.dll" (ByVal pBitmap()
_
      As Byte)
    Friend Declare Sub YUV422toRGBVB Lib "Camera.dll" (ByVal pBitmap()
_
      As Byte)
    Friend Declare Sub YUV444toRGBVB Lib "Camera.dll" (ByVal pBitmap()
_
      As Byte)
    Friend Declare Sub RGB16TORGBVB Lib "Camera.dll" (ByVal pBitmap() _
      As Byte)
    Friend Declare Function SetWBOnePush Lib "Camera.dll" () As Integer

    ' Camera Control Functions
    Friend Declare Sub InquireControlRegistersVB Lib "Camera.dll" ()
    Friend Declare Sub StatusControlRegistersVB Lib "Camera.dll" ()
    Friend Declare Sub SetBrightnessVB Lib "Camera.dll" ( _
      ByVal value As Integer)
    Friend Declare Sub SetAutoExposureVB Lib "Camera.dll" ( _
      ByVal value As Integer)
    Friend Declare Sub SetSharpnessVB Lib "Camera.dll" ( _
      ByVal value As Integer)
    Friend Declare Sub SetWhiteBalanceVB Lib "Camera.dll" ( _
      ByVal u As Integer, _
      ByVal v As Integer)
    Friend Declare Sub SetHueVB Lib "Camera.dll" (ByVal value As
Integer)
    Friend Declare Sub SetSaturationVB Lib "Camera.dll" ( _
      ByVal value As Integer)
    Friend Declare Sub SetGammaVB Lib "Camera.dll" ( _
      ByVal value As Integer)
    Friend Declare Sub SetShutterVB Lib "Camera.dll" ( _
      ByVal value As Integer)
    Friend Declare Sub SetGainVB Lib "Camera.dll" (ByVal value As
Integer)
    Friend Declare Sub SetIrisVB Lib "Camera.dll" (ByVal value As
Integer)
    Friend Declare Sub SetFocusVB Lib "Camera.dll" ( _
      ByVal value As Integer)
    Friend Declare Sub SetZoomVB Lib "Camera.dll" (ByVal value As
Integer)
```

```
    ' Camera Control Methods
    Friend Declare Function GetBrightnessMin Lib "Camera.dll" () As
Short
    Friend Declare Function GetBrightnessMax Lib "Camera.dll" () As
Short
    Friend Declare Function GetBrightnessValue1 Lib "Camera.dll" () _
      As Short
    Friend Declare Function GetAutoExposureMin Lib "Camera.dll" () _
      As Short
    Friend Declare Function GetAutoExposureMax Lib "Camera.dll" () _
      As Short
    Friend Declare Function GetAutoExposureValue1 Lib "Camera.dll" () _
      As Short
    Friend Declare Function GetSharpnessMin Lib "Camera.dll" () As
Short
    Friend Declare Function GetSharpnessMax Lib "Camera.dll" () As
Short
    Friend Declare Function GetSharpnessValue1 Lib "Camera.dll" () _
      As Short
    Friend Declare Function GetWBMin Lib "Camera.dll" () As Short
    Friend Declare Function GetWBMax Lib "Camera.dll" () As Short
    Friend Declare Function GetWBValue1 Lib "Camera.dll" () As Short
    Friend Declare Function GetWBValue2 Lib "Camera.dll" () As Short
    Friend Declare Function GetWBOnePushStatus Lib "Camera.dll" () As
Byte
    Friend Declare Function GetHueMin Lib "Camera.dll" () As Short
    Friend Declare Function GetHueMax Lib "Camera.dll" () As Short
    Friend Declare Function GetHueValue1 Lib "Camera.dll" () As Short
    Friend Declare Function GetSaturationMin Lib "Camera.dll" () As
Short
    Friend Declare Function GetSaturationMax Lib "Camera.dll" () As
Short
    Friend Declare Function GetSaturationValue1 Lib "Camera.dll" () _
      As Short
    Friend Declare Function GetGammaMin Lib "Camera.dll" () As Short
    Friend Declare Function GetGammaMax Lib "Camera.dll" () As Short
    Friend Declare Function GetGammaValue1 Lib "Camera.dll" () As Short
    Friend Declare Function GetShutterMin Lib "Camera.dll" () As Short
    Friend Declare Function GetShutterMax Lib "Camera.dll" () As Short
    Friend Declare Function GetShutterValue1 Lib "Camera.dll" () As
Short
    Friend Declare Function GetGainMin Lib "Camera.dll" () As Short
    Friend Declare Function GetGainMax Lib "Camera.dll" () As Short
    Friend Declare Function GetGainValue1 Lib "Camera.dll" () As Short
    Friend Declare Function GetIrisMin Lib "Camera.dll" () As Short
    Friend Declare Function GetIrisMax Lib "Camera.dll" () As Short
    Friend Declare Function GetIrisValue1 Lib "Camera.dll" () As Short
    Friend Declare Function GetFocusMin Lib "Camera.dll" () As Short
    Friend Declare Function GetFocusMax Lib "Camera.dll" () As Short
    Friend Declare Function GetFocusValue1 Lib "Camera.dll" () As Short
    Friend Declare Function GetZoomMin Lib "Camera.dll" () As Short
    Friend Declare Function GetZoomMax Lib "Camera.dll" () As Short
    Friend Declare Function GetZoomValue1 Lib "Camera.dll" () As Short
```

```
    ' Camera Control Functions
    Friend Declare Function SetWBOnePush Lib "Camera.dll" ( _
      ByVal value As Integer _
      ) As Integer

    'Function definitions for the C1394CameraControlSize Class Methods
    Friend Declare Function ControlSizemaxV Lib "Camera.dll" () As Byte
    Friend Declare Function ControlSizemaxH Lib "Camera.dll" () As Byte
    Friend Declare Function ControlSizeunitV Lib "Camera.dll" () As
Byte
    Friend Declare Function ControlSizeunitH Lib "Camera.dll" () As
Byte
    Friend Declare Function ControlSizeunitVpos Lib "Camera.dll" () _
      As Byte
    Friend Declare Function ControlSizeunitHpos Lib "Camera.dll" () _
      As Byte
    Friend Declare Function ControlSizetop Lib "Camera.dll" () As Byte
    Friend Declare Function ControlSizeleft Lib "Camera.dll" () As Byte
    Friend Declare Function ControlSizeheight Lib "Camera.dll" () As
Byte
    Friend Declare Function ControlSizewidth Lib "Camera.dll" () As
Byte
    Friend Declare Function ControlSizecolorCode Lib "Camera.dll" () _
      As Byte
    Friend Declare Function ControlSizepixelsFrame Lib "Camera.dll" ()
_
      As Byte
    Friend Declare Function ControlSizebytesframeHigh Lib "Camera.dll"
( _
      ) As Byte
    Friend Declare Function ControlSizebytesFrameLow Lib "Camera.dll" (
_
      ) As Byte
    Friend Declare Function ControlSizebytesPacketMin Lib "Camera.dll"
( _
      ) As Byte
    Friend Declare Function ControlSizebytesPacketMax Lib "Camera.dll"
( _
      ) As Byte
    Friend Declare Function ControlSizebytesPacket Lib "Camera.dll" ( _
      ) As Byte
    Friend Declare Function ControlSizepacketsFrame Lib "Camera.dll" (
_
      ) As Byte

    ' Function definitions for the C1394CameraControlSize Class
Functions
    Friend Declare Function SupportedVB Lib "Camera.dll" () As Byte
    Friend Declare Function ModeSupportedVB Lib "Camera.dll" () As Byte
    Friend Declare Function SetColorCodeVB Lib "Camera.dll" ( _
      ByVal code As Integer _
      ) As Integer
    Friend Declare Function SetSizeVB Lib "Camera.dll" ( _
```

```
      ByVal width As Integer, _
      ByVal height As Integer _
      ) As Integer
    Friend Declare Function SetPositionVB Lib "Camera.dll" ( _
      ByVal left As Integer, _
      ByVal top As Integer _
      ) As Integer
    Friend Declare Function SetBytesPerPacketVB Lib "Camera.dll" ( _
      ByVal bytes As Integer _
      ) As Integer
    Friend Declare Function InquireSizeVB Lib "Camera.dll" () As
Integer
    Friend Declare Function StatusSizeVB Lib "Camera.dll" () As Integer

    ' Function definitions for the C1394CameraControlTrigger Class
Members
    Friend Declare Function ControlTriggerpresent Lib "Camera.dll" ( _
      ) As Byte
    Friend Declare Function ControlTriggerreadout Lib "Camera.dll" ( _
      ) As Byte
    Friend Declare Function ControlTriggeronoff Lib "Camera.dll" ( _
      ) As Byte
    Friend Declare Function ControlTriggerpolarity Lib "Camera.dll" ( _
      ) As Byte
    'Friend Declare Function ControlTriggermode Lib "Camera.dll" ( _
      ) As Byte  'This is type pointer to byte
    Friend Declare Function ControlTriggerstatusPolarity Lib  _
      "Camera.dll" () As Byte
    Friend Declare Function ControlTriggerstatusOnOff Lib "Camera.dll"
(
      ) As Byte
    Friend Declare Function ControlTriggerstatusMode Lib "Camera.dll" (
_
      ) As Byte

    ' Function definitions for the C1394CameraControlTrigger Class
      Functions
    Friend Declare Function SetTriggerModeVB Lib "Camera.dll" ( _
      ByVal mode As Integer, _
      ByVal parameter As Integer _
      ) As Integer
    Friend Declare Function SetTriggerPolarityVB Lib "Camera.dll" ( _
      ByVal polarity As Byte _
      ) As Integer
    Friend Declare Function TurnTriggerOnVB Lib "Camera.dll" ( _
      ByVal triggerOn As Boolean _
      ) As Integer
    Friend Declare Function StatusTriggerVB Lib "Camera.dll" () As
Integer
    Friend Declare Function InquireTriggerVB Lib "Camera.dll" ( _
      ) As Integer

    Friend Declare Function InitializeRoutine Lib "Camera.dll" ( _
      ) As Integer
```

```
End Module
```

APPENDIX D

CAMERA DIALOG BOX SOURCE CODE

```
Imports System.Math

Public Class PicForm
    Inherits System.Windows.Forms.Form

#Region " Windows Form Designer generated code "

    Public Sub New()
        MyBase.New()

        'This call is required by the Windows Form Designer.
        InitializeComponent()

        'Add any initialization after the InitializeComponent() call

    End Sub

    'Form overrides dispose to clean up the component list.
    Protected Overloads Overrides Sub Dispose(ByVal disposing As
Boolean)
        If disposing Then
            If Not (components Is Nothing) Then
                components.Dispose()
            End If
        End If
        MyBase.Dispose(disposing)
    End Sub

    'Required by the Windows Form Designer
    Private components As System.ComponentModel.IContainer

    'NOTE: The following procedure is required by the Windows Form
Designer
    'It can be modified using the Windows Form Designer.
    'Do not modify it using the code editor.
    Friend WithEvents CMUImageBox As System.Windows.Forms.PictureBox
    Friend WithEvents PicTimer As System.Windows.Forms.Timer
    Friend WithEvents ControlsPanel As System.Windows.Forms.Panel
    Friend WithEvents TextBoxAutoExposure As
System.Windows.Forms.TextBox
    Friend WithEvents LabelAutoExposure As System.Windows.Forms.Label
    Friend WithEvents trbrAutoExposure As System.Windows.Forms.TrackBar
    Friend WithEvents TextBoxBrightness As System.Windows.Forms.TextBox
    Friend WithEvents TextBoxShutter As System.Windows.Forms.TextBox
    Friend WithEvents GroupBox1 As System.Windows.Forms.GroupBox
    Friend WithEvents LabelWBviolet As System.Windows.Forms.Label
    Friend WithEvents LabelWBblue As System.Windows.Forms.Label
    Friend WithEvents trbrWBblue As System.Windows.Forms.TrackBar
```

```
    Friend WithEvents trbrWBviolet As System.Windows.Forms.TrackBar
    Friend WithEvents TextBoxWBviolet As System.Windows.Forms.TextBox
    Friend WithEvents TextBoxWBblue As System.Windows.Forms.TextBox
    Friend WithEvents ButtonWBOnePush As System.Windows.Forms.Button
    Friend WithEvents LabelBrightness As System.Windows.Forms.Label
    Friend WithEvents LabelShutter As System.Windows.Forms.Label
    Friend WithEvents trbrBrightness As System.Windows.Forms.TrackBar
    Friend WithEvents trbrShutter As System.Windows.Forms.TrackBar
    Friend WithEvents HistoImageBox As System.Windows.Forms.PictureBox
    Friend WithEvents LeftBtn As System.Windows.Forms.CheckBox
    Friend WithEvents SetBtn As System.Windows.Forms.Button
    Friend WithEvents RightBtn As System.Windows.Forms.CheckBox
    Friend WithEvents PickLimitLbl As System.Windows.Forms.Label
    Friend WithEvents HistLbl As System.Windows.Forms.Label
    Friend WithEvents ApplyCloseBtn As System.Windows.Forms.Button
    <System.Diagnostics.DebuggerStepThrough()> Private Sub
InitializeComponent()
        Me.components = New System.ComponentModel.Container
        Dim resources As System.Resources.ResourceManager = New
System.Resources.ResourceManager(GetType(PicForm))
        Me.CMUImageBox = New System.Windows.Forms.PictureBox
        Me.PicTimer = New System.Windows.Forms.Timer(Me.components)
        Me.ControlsPanel = New System.Windows.Forms.Panel
        Me.HistLbl = New System.Windows.Forms.Label
        Me.HistoImageBox = New System.Windows.Forms.PictureBox
        Me.TextBoxAutoExposure = New System.Windows.Forms.TextBox
        Me.LabelAutoExposure = New System.Windows.Forms.Label
        Me.trbrAutoExposure = New System.Windows.Forms.TrackBar
        Me.TextBoxBrightness = New System.Windows.Forms.TextBox
        Me.TextBoxShutter = New System.Windows.Forms.TextBox
        Me.GroupBox1 = New System.Windows.Forms.GroupBox
        Me.LabelWBviolet = New System.Windows.Forms.Label
        Me.LabelWBblue = New System.Windows.Forms.Label
        Me.trbrWBblue = New System.Windows.Forms.TrackBar
        Me.trbrWBviolet = New System.Windows.Forms.TrackBar
        Me.TextBoxWBviolet = New System.Windows.Forms.TextBox
        Me.TextBoxWBblue = New System.Windows.Forms.TextBox
        Me.ButtonWBOnePush = New System.Windows.Forms.Button
        Me.LabelBrightness = New System.Windows.Forms.Label
        Me.LabelShutter = New System.Windows.Forms.Label
        Me.trbrBrightness = New System.Windows.Forms.TrackBar
        Me.trbrShutter = New System.Windows.Forms.TrackBar
        Me.SetBtn = New System.Windows.Forms.Button
        Me.RightBtn = New System.Windows.Forms.CheckBox
        Me.PickLimitLbl = New System.Windows.Forms.Label
        Me.LeftBtn = New System.Windows.Forms.CheckBox
        Me.ApplyCloseBtn = New System.Windows.Forms.Button
        Me.ControlsPanel.SuspendLayout()
        CType(Me.trbrAutoExposure,
System.ComponentModel.ISupportInitialize).BeginInit()
        Me.GroupBox1.SuspendLayout()
        CType(Me.trbrWBblue,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.trbrWBviolet,
```

```
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.trbrBrightness,
System.ComponentModel.ISupportInitialize).BeginInit()
        CType(Me.trbrShutter,
System.ComponentModel.ISupportInitialize).BeginInit()
        Me.SuspendLayout()
        '
        'CMUImageBox
        '
        Me.CMUImageBox.Location = New System.Drawing.Point(0, 0)
        Me.CMUImageBox.Name = "CMUImageBox"
        Me.CMUImageBox.Size = New System.Drawing.Size(240, 224)
        Me.CMUImageBox.SizeMode =
System.Windows.Forms.PictureBoxSizeMode.AutoSize
        Me.CMUImageBox.TabIndex = 1
        Me.CMUImageBox.TabStop = False
        '
        'PicTimer
        '
        '
        'ControlsPanel
        '
        Me.ControlsPanel.BorderStyle =
System.Windows.Forms.BorderStyle.Fixed3D
        Me.ControlsPanel.Controls.Add(Me.ApplyCloseBtn)
        Me.ControlsPanel.Controls.Add(Me.HistLbl)
        Me.ControlsPanel.Controls.Add(Me.HistoImageBox)
        Me.ControlsPanel.Controls.Add(Me.TextBoxAutoExposure)
        Me.ControlsPanel.Controls.Add(Me.LabelAutoExposure)
        Me.ControlsPanel.Controls.Add(Me.trbrAutoExposure)
        Me.ControlsPanel.Controls.Add(Me.TextBoxBrightness)
        Me.ControlsPanel.Controls.Add(Me.TextBoxShutter)
        Me.ControlsPanel.Controls.Add(Me.GroupBox1)
        Me.ControlsPanel.Controls.Add(Me.LabelBrightness)
        Me.ControlsPanel.Controls.Add(Me.LabelShutter)
        Me.ControlsPanel.Controls.Add(Me.trbrBrightness)
        Me.ControlsPanel.Controls.Add(Me.trbrShutter)
        Me.ControlsPanel.Controls.Add(Me.SetBtn)
        Me.ControlsPanel.Controls.Add(Me.RightBtn)
        Me.ControlsPanel.Controls.Add(Me.PickLimitLbl)
        Me.ControlsPanel.Controls.Add(Me.LeftBtn)
        Me.ControlsPanel.Location = New System.Drawing.Point(256, 8)
        Me.ControlsPanel.Name = "ControlsPanel"
        Me.ControlsPanel.Size = New System.Drawing.Size(272, 472)
        Me.ControlsPanel.TabIndex = 2
        '
        'HistLbl
        '
        Me.HistLbl.Font = New System.Drawing.Font("Microsoft Sans
Serif", 9.0!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.HistLbl.Location = New System.Drawing.Point(8, 339)
        Me.HistLbl.Name = "HistLbl"
        Me.HistLbl.Size = New System.Drawing.Size(256, 23)
```

```
        Me.HistLbl.TabIndex = 38
        Me.HistLbl.Text = "Intensity Histogram"
        Me.HistLbl.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'HistoImageBox
        '
        Me.HistoImageBox.Location = New System.Drawing.Point(8, 208)
        Me.HistoImageBox.Name = "HistoImageBox"
        Me.HistoImageBox.Size = New System.Drawing.Size(256, 128)
        Me.HistoImageBox.TabIndex = 37
        Me.HistoImageBox.TabStop = False
        '
        'TextBoxAutoExposure
        '
        Me.TextBoxAutoExposure.Enabled = False
        Me.TextBoxAutoExposure.Location = New System.Drawing.Point(61,
136)
        Me.TextBoxAutoExposure.Name = "TextBoxAutoExposure"
        Me.TextBoxAutoExposure.Size = New System.Drawing.Size(32, 20)
        Me.TextBoxAutoExposure.TabIndex = 36
        Me.TextBoxAutoExposure.Text = ""
        '
        'LabelAutoExposure
        '
        Me.LabelAutoExposure.Font = New System.Drawing.Font("Microsoft
Sans Serif", 7.5!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.LabelAutoExposure.Location = New System.Drawing.Point(56, 9)
        Me.LabelAutoExposure.Name = "LabelAutoExposure"
        Me.LabelAutoExposure.Size = New System.Drawing.Size(40, 23)
        Me.LabelAutoExposure.TabIndex = 35
        Me.LabelAutoExposure.Text = "Analog Gain"
        Me.LabelAutoExposure.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'trbrAutoExposure
        '
        Me.trbrAutoExposure.LargeChange = 1
        Me.trbrAutoExposure.Location = New System.Drawing.Point(56, 32)
        Me.trbrAutoExposure.Name = "trbrAutoExposure"
        Me.trbrAutoExposure.Orientation =
System.Windows.Forms.Orientation.Vertical
        Me.trbrAutoExposure.Size = New System.Drawing.Size(42, 104)
        Me.trbrAutoExposure.TabIndex = 34
        Me.trbrAutoExposure.TickStyle =
System.Windows.Forms.TickStyle.Both
        '
        'TextBoxBrightness
        '
        Me.TextBoxBrightness.Enabled = False
        Me.TextBoxBrightness.Location = New System.Drawing.Point(109,
136)
        Me.TextBoxBrightness.Name = "TextBoxBrightness"
```

```
        Me.TextBoxBrightness.Size = New System.Drawing.Size(32, 20)
        Me.TextBoxBrightness.TabIndex = 31
        Me.TextBoxBrightness.Text = ""
        '
        'TextBoxShutter
        '
        Me.TextBoxShutter.Enabled = False
        Me.TextBoxShutter.Location = New System.Drawing.Point(16, 136)
        Me.TextBoxShutter.Name = "TextBoxShutter"
        Me.TextBoxShutter.Size = New System.Drawing.Size(32, 20)
        Me.TextBoxShutter.TabIndex = 30
        Me.TextBoxShutter.Text = ""
        '
        'GroupBox1
        '
        Me.GroupBox1.Controls.Add(Me.LabelWBviolet)
        Me.GroupBox1.Controls.Add(Me.LabelWBblue)
        Me.GroupBox1.Controls.Add(Me.trbrWBblue)
        Me.GroupBox1.Controls.Add(Me.trbrWBviolet)
        Me.GroupBox1.Controls.Add(Me.TextBoxWBviolet)
        Me.GroupBox1.Controls.Add(Me.TextBoxWBblue)
        Me.GroupBox1.Controls.Add(Me.ButtonWBOnePush)
        Me.GroupBox1.Font = New System.Drawing.Font("Microsoft Sans
Serif", 7.5!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.GroupBox1.Location = New System.Drawing.Point(160, 8)
        Me.GroupBox1.Name = "GroupBox1"
        Me.GroupBox1.Size = New System.Drawing.Size(96, 192)
        Me.GroupBox1.TabIndex = 29
        Me.GroupBox1.TabStop = False
        Me.GroupBox1.Text = "White Balance"
        '
        'LabelWBviolet
        '
        Me.LabelWBviolet.Font = New System.Drawing.Font("Microsoft Sans
Serif", 7.5!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.LabelWBviolet.Location = New System.Drawing.Point(49, 16)
        Me.LabelWBviolet.Name = "LabelWBviolet"
        Me.LabelWBviolet.Size = New System.Drawing.Size(40, 23)
        Me.LabelWBviolet.TabIndex = 9
        Me.LabelWBviolet.Text = "v"
        Me.LabelWBviolet.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'LabelWBblue
        '
        Me.LabelWBblue.Font = New System.Drawing.Font("Microsoft Sans
Serif", 7.5!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.LabelWBblue.Location = New System.Drawing.Point(4, 16)
        Me.LabelWBblue.Name = "LabelWBblue"
        Me.LabelWBblue.Size = New System.Drawing.Size(40, 23)
        Me.LabelWBblue.TabIndex = 8
```

```
        Me.LabelWBblue.Text = "u"
        Me.LabelWBblue.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'trbrWBblue
        '
        Me.trbrWBblue.LargeChange = 1
        Me.trbrWBblue.Location = New System.Drawing.Point(3, 24)
        Me.trbrWBblue.Name = "trbrWBblue"
        Me.trbrWBblue.Orientation =
System.Windows.Forms.Orientation.Vertical
        Me.trbrWBblue.Size = New System.Drawing.Size(42, 104)
        Me.trbrWBblue.TabIndex = 2
        Me.trbrWBblue.TickStyle = System.Windows.Forms.TickStyle.Both
        '
        'trbrWBviolet
        '
        Me.trbrWBviolet.LargeChange = 1
        Me.trbrWBviolet.Location = New System.Drawing.Point(48, 24)
        Me.trbrWBviolet.Name = "trbrWBviolet"
        Me.trbrWBviolet.Orientation =
System.Windows.Forms.Orientation.Vertical
        Me.trbrWBviolet.Size = New System.Drawing.Size(42, 104)
        Me.trbrWBviolet.TabIndex = 3
        Me.trbrWBviolet.TickStyle = System.Windows.Forms.TickStyle.Both
        '
        'TextBoxWBviolet
        '
        Me.TextBoxWBviolet.Enabled = False
        Me.TextBoxWBviolet.Location = New System.Drawing.Point(53, 128)
        Me.TextBoxWBviolet.Name = "TextBoxWBviolet"
        Me.TextBoxWBviolet.Size = New System.Drawing.Size(32, 19)
        Me.TextBoxWBviolet.TabIndex = 17
        Me.TextBoxWBviolet.Text = ""
        '
        'TextBoxWBblue
        '
        Me.TextBoxWBblue.Enabled = False
        Me.TextBoxWBblue.Location = New System.Drawing.Point(8, 128)
        Me.TextBoxWBblue.Name = "TextBoxWBblue"
        Me.TextBoxWBblue.Size = New System.Drawing.Size(32, 19)
        Me.TextBoxWBblue.TabIndex = 16
        Me.TextBoxWBblue.Text = ""
        '
        'ButtonWBOnePush
        '
        Me.ButtonWBOnePush.Font = New System.Drawing.Font("Microsoft
Sans Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.ButtonWBOnePush.Location = New System.Drawing.Point(8, 152)
        Me.ButtonWBOnePush.Name = "ButtonWBOnePush"
        Me.ButtonWBOnePush.Size = New System.Drawing.Size(80, 32)
        Me.ButtonWBOnePush.TabIndex = 20
        Me.ButtonWBOnePush.Text = "Auto WB"
```

```
        '
        'LabelBrightness
        '
        Me.LabelBrightness.Font = New System.Drawing.Font("Microsoft
Sans Serif", 7.5!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.LabelBrightness.Location = New System.Drawing.Point(93, 9)
        Me.LabelBrightness.Name = "LabelBrightness"
        Me.LabelBrightness.Size = New System.Drawing.Size(64, 23)
        Me.LabelBrightness.TabIndex = 28
        Me.LabelBrightness.Text = "Brightness"
        Me.LabelBrightness.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'LabelShutter
        '
        Me.LabelShutter.Font = New System.Drawing.Font("Microsoft Sans
Serif", 7.5!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.LabelShutter.Location = New System.Drawing.Point(7, 9)
        Me.LabelShutter.Name = "LabelShutter"
        Me.LabelShutter.Size = New System.Drawing.Size(50, 23)
        Me.LabelShutter.TabIndex = 27
        Me.LabelShutter.Text = "Shutter"
        Me.LabelShutter.TextAlign =
System.Drawing.ContentAlignment.MiddleCenter
        '
        'trbrBrightness
        '
        Me.trbrBrightness.LargeChange = 1
        Me.trbrBrightness.Location = New System.Drawing.Point(104, 32)
        Me.trbrBrightness.Name = "trbrBrightness"
        Me.trbrBrightness.Orientation =
System.Windows.Forms.Orientation.Vertical
        Me.trbrBrightness.Size = New System.Drawing.Size(42, 104)
        Me.trbrBrightness.TabIndex = 26
        Me.trbrBrightness.TickStyle =
System.Windows.Forms.TickStyle.Both
        '
        'trbrShutter
        '
        Me.trbrShutter.LargeChange = 1
        Me.trbrShutter.Location = New System.Drawing.Point(11, 32)
        Me.trbrShutter.Name = "trbrShutter"
        Me.trbrShutter.Orientation =
System.Windows.Forms.Orientation.Vertical
        Me.trbrShutter.Size = New System.Drawing.Size(42, 104)
        Me.trbrShutter.TabIndex = 25
        Me.trbrShutter.TickStyle = System.Windows.Forms.TickStyle.Both
        '
        'SetBtn
        '
        Me.SetBtn.BackColor = System.Drawing.SystemColors.Control
        Me.SetBtn.Font = New System.Drawing.Font("Microsoft Sans
```

```
Serif", 14.0!, System.Drawing.FontStyle.Regular,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.SetBtn.Location = New System.Drawing.Point(88, 416)
        Me.SetBtn.Name = "SetBtn"
        Me.SetBtn.Size = New System.Drawing.Size(96, 40)
        Me.SetBtn.TabIndex = 8
        Me.SetBtn.Text = "Set"
        '
        'RightBtn
        '
        Me.RightBtn.Appearance = System.Windows.Forms.Appearance.Button
        Me.RightBtn.Image =
CType(resources.GetObject("RightBtn.Image"), System.Drawing.Image)
        Me.RightBtn.Location = New System.Drawing.Point(200, 416)
        Me.RightBtn.Name = "RightBtn"
        Me.RightBtn.Size = New System.Drawing.Size(48, 40)
        Me.RightBtn.TabIndex = 9
        '
        'PickLimitLbl
        '
        Me.PickLimitLbl.Font = New System.Drawing.Font("Microsoft Sans
Serif", 16.0!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.PickLimitLbl.Location = New System.Drawing.Point(32, 376)
        Me.PickLimitLbl.Name = "PickLimitLbl"
        Me.PickLimitLbl.Size = New System.Drawing.Size(208, 32)
        Me.PickLimitLbl.TabIndex = 10
        Me.PickLimitLbl.Text = "Pick 1st Az Limit"
        Me.PickLimitLbl.TextAlign =
System.Drawing.ContentAlignment.TopCenter
        '
        'LeftBtn
        '
        Me.LeftBtn.Appearance = System.Windows.Forms.Appearance.Button
        Me.LeftBtn.Image = CType(resources.GetObject("LeftBtn.Image"),
System.Drawing.Image)
        Me.LeftBtn.Location = New System.Drawing.Point(24, 416)
        Me.LeftBtn.Name = "LeftBtn"
        Me.LeftBtn.Size = New System.Drawing.Size(48, 40)
        Me.LeftBtn.TabIndex = 7
        '
        'ApplyCloseBtn
        '
        Me.ApplyCloseBtn.Font = New System.Drawing.Font("Microsoft Sans
Serif", 8.25!, System.Drawing.FontStyle.Bold,
System.Drawing.GraphicsUnit.Point, CType(0, Byte))
        Me.ApplyCloseBtn.Location = New System.Drawing.Point(40, 163)
        Me.ApplyCloseBtn.Name = "ApplyCloseBtn"
        Me.ApplyCloseBtn.Size = New System.Drawing.Size(80, 32)
        Me.ApplyCloseBtn.TabIndex = 39
        Me.ApplyCloseBtn.Text = "Apply && Close"
        '
        'PicForm
        '
```

```
        Me.AutoScaleBaseSize = New System.Drawing.Size(5, 13)
        Me.ClientSize = New System.Drawing.Size(530, 495)
        Me.ControlBox = False
        Me.Controls.Add(Me.ControlsPanel)
        Me.Controls.Add(Me.CMUImageBox)
        Me.FormBorderStyle =
System.Windows.Forms.FormBorderStyle.FixedSingle
        Me.Name = "PicForm"
        Me.ShowInTaskbar = False
        Me.Text = "Camera Image"
        Me.ControlsPanel.ResumeLayout(False)
        CType(Me.trbrAutoExposure,
System.ComponentModel.ISupportInitialize).EndInit()
        Me.GroupBox1.ResumeLayout(False)
        CType(Me.trbrWBblue,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.trbrWBviolet,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.trbrBrightness,
System.ComponentModel.ISupportInitialize).EndInit()
        CType(Me.trbrShutter,
System.ComponentModel.ISupportInitialize).EndInit()
        Me.ResumeLayout(False)

    End Sub

#End Region

#Region " Variables Global to PicForm"
    Private Const XDimension As Integer = 768           'X Dimension of
Viewable Image
    Private Const YDimension As Integer = 2048          'Y Dimension of
Viewable Image
    Private Const XLimit As Integer = 1536 - XDimension 'X Dimension
Position Limit
    Private Const XYRatio As Integer = 4                'Ratio of
Viewable Image
    Private pbitmap(XDimension * YDimension * 3 - 1) As Byte    'Buffer
for Image
    Private cmubmp As New Bitmap(CInt(XDimension / XYRatio),
CInt(YDimension / XYRatio), Imaging.PixelFormat.Format24bppRgb) 'Image
Bitmap
    'Private isloaded As Boolean = False
    Private iserror As Boolean = False                  'Shows error
    Private histpic As New Bitmap(256, 128)             'Histogram
Bitmap
    Private bins(255) As Integer                        'Binning for
Histogram
    Private XPos As Integer = 0                         'Position of
Viewable Image
#End Region

#Region " Initialization and Closing "
    Private Sub PicForm_Load(ByVal sender As System.Object, ByVal e As
```

```
System.EventArgs) Handles MyBase.Load
        'Size form according to image size
        Me.Height = CMUImageBox.Height + 15
        Me.Width = CMUImageBox.Width + 10

        ' Initialize Camera
        If Not CheckLinkVB() = CAM_SUCCESS Then
            MsgBox("Error: No camera found")
            End
        End If

        If Not InitCameraVB() = CAM_SUCCESS Then
            MsgBox("Error: No camera initialized")
            End
        End If

        InquireControlRegistersVB()
        StatusControlRegistersVB()
        SupportedVB()
        InquireSizeVB()
        StatusSizeVB()

        If Not SetVideoFormatVB(Convert.ToUInt32(7)) = CAM_SUCCESS Then
            MsgBox("Error: Unable to set video format")
            End
        End If

        If Not SetVideoModeVB(Convert.ToUInt32(0)) = CAM_SUCCESS Then
            MsgBox("Error: Unable to set video mode")
            End
        End If

        If Not SetColorCodeVB(4) = CAM_SUCCESS Then
            MsgBox("Error: Unable to set color code")
            End
        End If

        'Make variable x, y dependant on setup settings
        If Not SetSizeVB(YDimension, XDimension) = CAM_SUCCESS Then
            MsgBox("Error: Unable to set video size")
            End
        End If

        'Make variable position dependant on setup settings
        'camerawidth
        If CameraLeft > XLimit Then
            XPos = XLimit
        Else
            XPos = CameraLeft + CameraWidth / 2 - XDimension / 2
            If XPos < 0 Then XPos = 0
        End If

        If Not SetPositionVB(0, XPos) = CAM_SUCCESS Then
            MsgBox("Error: Unable to set video position")
```

```
            End
        End If

        If Not SetBytesPerPacketVB(4096) = CAM_SUCCESS Then
            MsgBox("Error: Unable to set packet size")
            End
        End If

        If Not StartImageCaptureVB() = CAM_SUCCESS Then
            StopImageCaptureVB()
            TurnTriggerOnVB(False)
            MsgBox("Error: Unable to set image capture")
            End
        End If

        ' Initialize Camera Controls
        Dim checkshutter As Integer
        Dim checkautoexposure As Integer

        InquireControlRegistersVB()
        StatusControlRegistersVB()

        ' Shutter Initialization
        trbrShutter.Minimum = GetShutterMin
        trbrShutter.Maximum = 44 'This is the largest time that can
still be triggered at 60Hz
        trbrShutter.TickFrequency = (trbrShutter.Maximum -
trbrShutter.Minimum) / 10
        If CameraShutterSpeed > 44 Then CameraShutterSpeed = 44
        If CameraShutterSpeed < trbrShutter.Minimum Then
CameraShutterSpeed = trbrShutter.Minimum
        SetShutterVB(CameraShutterSpeed)
        trbrShutter.Value = GetShutterValue1
        TextBoxShutter.Text = GetShutterValue1

        ' Auto Exposure Initialization
        trbrAutoExposure.Minimum = GetAutoExposureMin
        trbrAutoExposure.Maximum = GetAutoExposureMax 'Reduced slightly
so the tick marks look good
        trbrAutoExposure.TickFrequency = (trbrAutoExposure.Maximum -
trbrAutoExposure.Minimum - (trbrAutoExposure.Maximum - 100)) / 10
        If CameraAnalogGain > trbrAutoExposure.Maximum Then
CameraAnalogGain = trbrAutoExposure.Maximum
        If CameraAnalogGain < trbrAutoExposure.Minimum Then
CameraAnalogGain = trbrAutoExposure.Minimum
        SetAutoExposureVB(CameraAnalogGain)
        trbrAutoExposure.Value = GetAutoExposureValue1
        TextBoxAutoExposure.Text = GetAutoExposureValue1

        ' White Balance Initialization
        trbrWBblue.Minimum = GetWBMin
        trbrWBblue.Maximum = GetWBMax
        trbrWBblue.TickFrequency = (trbrWBblue.Maximum -
trbrWBblue.Minimum) / 10
```

```
        trbrWBblue.Value = GetWBValue2
        trbrWBviolet.Minimum = GetWBMin
        trbrWBviolet.Maximum = GetWBMax
        trbrWBviolet.TickFrequency = (trbrWBviolet.Maximum -
trbrWBviolet.Minimum) / 10
        trbrWBviolet.Value = GetWBValue1
        TextBoxWBblue.Text = GetWBValue2
        TextBoxWBviolet.Text = GetWBValue1

        ' Brightness Initialization
        trbrBrightness.Minimum = GetBrightnessMin
        trbrBrightness.Maximum = GetBrightnessMax
        trbrBrightness.TickFrequency = (trbrBrightness.Maximum -
trbrBrightness.Minimum) / 10
        If CameraBrightness > trbrBrightness.Maximum Then
CameraBrightness = trbrBrightness.Maximum
        If CameraBrightness < trbrBrightness.Minimum Then
CameraBrightness = trbrBrightness.Minimum
        SetBrightnessVB(CameraBrightness)
        trbrBrightness.Value = GetBrightnessMax - CameraBrightness
        TextBoxBrightness.Text = GetBrightnessValue1

        ' Digital Gain Initialization
        SetGainVB(GetGainMin) 'sets digital gain to 1

        CMUImageBox.Image = cmubmp
        HistoImageBox.Image = histpic
        PicTimer.Enabled = True
        'isloaded = True
    End Sub

    Private Sub PicForm_Closing(ByVal sender As System.Object, ByVal e
As System.ComponentModel.CancelEventArgs) Handles MyBase.Closing
        KillImage()
    End Sub

    Friend Sub KillImage()
        PicTimer.Enabled = False
        Try
            StopImageCaptureVB()
            TurnTriggerOnVB(False)
        Catch
        End Try
    End Sub

    'Resize form according to image size and place controls
appropriately
    Private Sub CMUImageBox_Resize(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CMUImageBox.Resize
        Me.Width = CMUImageBox.Width + ControlsPanel.Width + 20
        CMUImageBox.Location = New Point(5, 5)
        ControlsPanel.Location = New Point(CMUImageBox.Width + 10, 5)

        If CMUImageBox.Height > ControlsPanel.Height Then
```

```
            Me.Height = CMUImageBox.Height + 40
        Else
            Me.Height = ControlsPanel.Height + 40
        End If
    End Sub
#End Region

#Region " Image Refresh Timer "
    Private Sub PicTimer_Tick(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles PicTimer.Tick
        'Try Campturing Image
        If Not CaptureImageVB() = CAM_SUCCESS Then
            StopImageCaptureVB()
            TurnTriggerOnVB(False)
            If Not iserror Then
                MsgBox("The image could not display")
                iserror = True
            End If
            Exit Sub
        End If

        iserror = False
        getRGBVB(pbitmap)

        Dim i, j, k, n, m As Integer
        Dim maxbin As Integer = 0

        Dim binner As Double
        Dim centerY, centerX, halfwidthcross As Integer
        centerY = YDimension / (2 * XYRatio)
        centerX = (CameraLeft + CameraWidth / 2) / XYRatio - XPos /
XYRatio
        halfwidthcross = CameraWidth / (XYRatio * 2)
        Array.Clear(bins, 0, bins.Length)
        k = 0
        For j = 0 To CInt(XDimension / XYRatio) - 1 Step 1
            For i = CInt(YDimension / XYRatio) - 1 To 0 Step -1
                'Place crosshair or image
                If i <= centerY + 10 AndAlso i > centerY - 10 AndAlso j
_
                  <= centerX + halfwidthcross AndAlso j > centerX - _
                  halfwidthcross Then
                    cmubmp.SetPixel(j, i, Color.Red)
                ElseIf j <= centerX + 10 AndAlso j > centerX - 10 _
                  AndAlso i <= centerY + halfwidthcross AndAlso i > _
                  centerY - halfwidthcross Then
                    cmubmp.SetPixel(j, i, Color.Red)
                Else
                    cmubmp.SetPixel(j, i, Color.FromArgb(pbitmap(k) _
                        , pbitmap(k + 1), pbitmap(k + 2)))
                End If

                'Bin Intensity Image for Histogram -- Use Y image in
YUV
```

```
                'translation from RGB
                binner = CDbl(pbitmap(k)) * 0.299 + CDbl(pbitmap(k +
1)) _
                * 0.587 + CDbl(pbitmap(k + 2)) * 0.114
                If binner > 255 Then binner = 255
                bins(CInt(binner)) += 1
                k += XYRatio * 3
            Next i
            k += YDimension * 3 * (XYRatio - 1)
        Next j

        'Create a histogram
        For n = 0 To 255
            For m = 0 To 127
                histpic.SetPixel(n, m, Color.White)
            Next m
        Next n

        For m = 0 To 255
            maxbin = Max(bins(m), maxbin)
        Next m

        For m = 0 To 255
            k = CInt(bins(m) / (maxbin / 128))
            For n = 0 To k - 1
                histpic.SetPixel(m, 127 - n, Color.Black)
            Next n
        Next m

        CMUImageBox.Refresh()
        HistoImageBox.Refresh()
    End Sub
#End Region

#Region " Camera controls "
    ' These functions handle the trackbar scroll commands
    Private Sub trbrShutter_Scroll( _
ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
trbrShutter.Scroll
        CameraShutterSpeed = trbrShutter.Value
        SetShutterVB(CameraShutterSpeed)
        TextBoxShutter.Text = GetShutterValue1
    End Sub

    Private Sub trbrAutoExposure_Scroll(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles trbrAutoExposure.Scroll
        CameraAnalogGain = trbrAutoExposure.Value
        SetAutoExposureVB(CameraAnalogGain)
        TextBoxAutoExposure.Text = GetAutoExposureValue1
    End Sub

    Private Sub trbrBrightness_Scroll(ByVal sender As Object, ByVal e
As System.EventArgs) Handles trbrBrightness.Scroll
        CameraBrightness = GetBrightnessMax - trbrBrightness.Value
```

```
        SetBrightnessVB(CameraBrightness)
        TextBoxBrightness.Text = GetBrightnessValue1
    End Sub

    Private Sub trbrWBblue_Scroll(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles trbrWBblue.Scroll
        SetWhiteBalanceVB(trbrWBblue.Value, trbrWBviolet.Value)
        TextBoxWBblue.Text = GetWBValue2
        TextBoxWBviolet.Text = GetWBValue1
    End Sub

    Private Sub trbrWBviolet_Scroll(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles trbrWBviolet.Scroll
        SetWhiteBalanceVB(trbrWBblue.Value, trbrWBviolet.Value)
        TextBoxWBblue.Text = GetWBValue2
        TextBoxWBviolet.Text = GetWBValue1
    End Sub

    Private Sub ButtonWBOnePush_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ButtonWBOnePush.Click
        SetWBOnePush()
        InquireControlRegistersVB()
        StatusControlRegistersVB()
        ' White Balance Initialization
        trbrWBblue.Minimum = GetWBMin
        trbrWBblue.Maximum = GetWBMax
        trbrWBblue.TickFrequency = (trbrWBblue.Maximum -
trbrWBblue.Minimum) / 10
        trbrWBblue.Value = GetWBValue2
        trbrWBviolet.Minimum = GetWBMin
        trbrWBviolet.Maximum = GetWBMax
        trbrWBviolet.TickFrequency = (trbrWBviolet.Maximum -
trbrWBviolet.Minimum) / 10
        trbrWBviolet.Value = GetWBValue1
        TextBoxWBblue.Text = GetWBValue2
        TextBoxWBviolet.Text = GetWBValue1
    End Sub
#End Region

#Region "LIDAR Scanner Controls"
    'Handles Right Push Button (Uses Scan_Console objects)
    Private Sub RightBtn_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles RightBtn.Click
        ScanForm.RightBtn.Checked = Not ScanForm.RightBtn.Checked
        ScanForm.RightBtnAction()
    End Sub

    'Handles Left Push Button (Uses Scan_Console objects)
    Private Sub LeftBtn_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles LeftBtn.Click
        ScanForm.LeftBtn.Checked = Not ScanForm.LeftBtn.Checked
        ScanForm.LeftBtnAction()
    End Sub
```

```
    'Handles Set Push Button (Uses Scan_Console objects)
    Private Sub SetBtn_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles SetBtn.Click
        ScanForm.SetBtn.PerformClick()
    End Sub

    'Handles Clicking of Apply and Close Button (Uses Scan_Console
objects)
    Private Sub ApplyCloseBtn_Click(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles ApplyCloseBtn.Click
        ScanForm.PanelSetup.Visible = True
        ScanForm.PanelPickLim.Visible = False
        ScanForm.PickLimBtn.Text = "Pick Limits"
        ScanForm.PickLimitLbl.Text = "Pick 1st Az Limit"
        PickLimitLbl.Text = "Pick 1st Az Limit"
        Try
            Me.KillImage()
            Me.Dispose()
        Catch
        Finally
            ScanForm.Refresh()
        End Try
    End Sub
#End Region

End Class
```

APPENDIX E

CAIL TEX FILE FORMAT DEFINITION

CAIL TEX file format

The TEX file format was established during the design of Prototype I. The original TEX file format is a collection of RAW 8 bit (64 (h) x 20 (v) pixel) images written to file sequentially. Since there is no header attached to this file, there is no way to distinguish TEX files generated by Prototype I from subsequent TEX files. Therefore, a header is added to the TEX file in Prototype II: Version 1.0.0. This provides adequate information about the images captured within the file. To allow for backwards compatibility this formal document was created. The following document contains the existing TEX file information as well as the proposed TEX file format.

Prototype I: Version 0.0.0

Structure:

Images

Field #	Data	Bytes	Format	Units	Offset	Description
1	Images	Varies	RAW8 64 x 20 pixels/image	-	0	Images in specified format

Prototype II: Version 1.0.0

Structure:

Header	Images

Field #	Data	Bytes	Format	Units	Offset	Description
1 (header)	Header Size	4	integer	bytes	0	Number of bytes in header
	Width	4	integer	pixels	4	Width of image
	Height	4	integer	pixels	8	Height of image
	Time	4	integer	microsecs	12	Period between image captures
2	Images	Varies	RGB 3 Bytes/Pixel	-	16	Images in specified format

PROTOTYPE II: VERSION 1.1.0

Header	Images

Field #	Data	Bytes	Format	Units	Offset	Description
1 (header)	Header Size	4	integer	bytes	0	Number of bytes in header
	Data Offset	4	integer	bytes	4	Number of bytes to data
	Version: Major	4	integer		8	Major
	Version: Minor	4	integer		12	Minor
	Version: Update	4	integer		16	Update
	Date/Time	8	double		20	When the file was originally created
(EO info)	Camera Serial Number	4	integer		28	
	Pixel Height	8	double	µm	32	physical pixel dimensions
	Pixel Width	8	double	µm	40	
	Max Height	4	integer	pixels	48	Number of pixels in height
	Max Width	4	integer	pixels	52	Number of pixels in width
	Height	4	integer	pixels	56	Height of image
	Width	4	integer	pixels	60	Width of image
	Array Offset Left	4	integer	pixels	64	image offset on array
	Array Offset Bottom	4	integer	pixels	68	image offset on array
	Image Format (needs changing)	1	byte	From table	72	Pixel data type in image
	Image Scalar (needs changing)	1	byte	From Table	73	Number of bytes per pixel
	Focal Length	4	double	mm	74	Lens focal length
	Camera Offset X	8	double	m	78	Physical offset from lidar

	Camera Offset Y	8	double	m	86	centroid (parallax)
	Camera Offset Z	8	double	m	94	
	Camera Rotation X	8	double	deg	102	Physical rotation offset of camera from lidar centroid
	Camera Rotation Y	8	double	deg	110	
	Camera Rotation Z	8	double	deg	118	
	Camera Gain	8	double		126	relative value
	Camera Brightness	8	double		124	
	Camera Exposure	8	double		132	
	Camera WB violet	8	double		140	
	Camera WB blue	8	double		148	
(lidar info)	Lidar Serial Number	8	char		156	
	Az/Hfov Left	8	double	deg	164	
	Az/Hfov Right	8	double	deg	172	
	Elevation/ Vfov Top	8	double	deg	180	
	Elevation/ Vfov Bottom	8	double	deg	188	
	tilt angle	8	double	deg	196	angle of texel camera tilt through tilt axis
	tilt offset	8	double		204	??
	tilt axis	8	double		212	tilt axis
(GPS info)	GPS Serial Number	8	char		220	
	GPS Offset X	8	double	m	228	distance from lidar centroid to antenna
	GPS Offset Y	8	double	m	236	
	GPS Offset Z	8	double	m	244	
	Latitude/ Northing	8	double	deg	252	

	Longitude/ Easting	8	double	deg	260	
	Azimuth/ Elevation	8	double	m	268	
	Zone (type)	4	integer		276	Indicates lat/long or N,E and Azimuth
	Projection	4	double		284	
	Datum	8	char		292	
2	Images	Varies	= byte * Image Scalar		300	Images in specified format

Image Format Value	Image Format	Image Scalar
0	RAW8	1
1	RAW10	2
2	RGB	3
3	YUV	3
4	Monochrome	1
>=3	Undefined in Vs. 1.1.0	Undefined in Vs 1.1.0