A System to Redirect Solar Energy for Wireless Power Transmission

Luc Bettaieb, Chris Goodeaux, Vadim Linevich, and Andrew Powell

Marshall Space Flight Center

August 9, 2013

Reviewed by NASA Mentor
Dean Alhorn
ES32 – Control and Signal Conditioning Electronics Branch

____
Mentor Signature Here

A System to Redirect Solar Energy for Wireless Power Transmission

Luc Bettaieb[1]
Case Western Reserve University, Cleveland, OH, 44106

Christopher Goodeaux[2]
The University of Alabama, Tuscaloosa, AL, 35401

and

Vadim Linevich[3] and Andrew Powell[4]
Temple University, Philadelphia, PA, 19122

Lack of a prolonged energy source during exploration of dark craters on the lunar surface is a problem that could be solved several ways. Our team determined that an ideal solution for this predicament would not be to tether, but to directly beam solar energy at the target rover. This system, tentatively known as Solarbeam, would not only be able to autonomously track the sun using sun sensors, but also be able to use an algorithm to determine the position of the rover and beam the energy directly to its solar panels. Currently, a prototype system for use in proof of concept testing is being developed and aims to produce results that will make way for future development to begin.

I. Introduction

Research into the wireless transmission of power has grown quite steadily in recent years. This has been spurred on by the current limits of energy storage systems and batteries, like limited energy density. This research carries even more importance in the field of space systems, where mass is at a premium. With a working wireless transmission system, space systems could allocate more of their mass towards higher-precision instruments instead of power systems.

A research team under Principal Investigator Dean Alhorn began research into a wireless transmission system using simple mirrors to redirect sunlight. A system of mirrors could beam sunlight to an array of solar panels, even if the array was not in direct view of the sun. A test scenario using the Shackleton impact crater on the lunar South Pole was developed to flesh out our concept. Research into the crater revealed that spots along the rim of the crater received near-constant illumination throughout the year, while the inside of the crater resides in permanent shadow (Bussey et al., 2010). Our goals were to develop a test article that could redirect low-incidence angles of sunlight to a shadowed target, and autonomously track both the sun and a moving target.

II. Design

In creating a design for a test article, the research team evaluated many potential designs. Eventually, three features were determined as essential to the concept. A primary *collector mirror* would autonomously track the sun, on one degree-of-freedom, and redirect the rays to a secondary *relay mirror.* This relay would then bounce the sunlight to a designated target. Lastly, a *mast* would be used to elevate the system to increase the transmission range of sunlight.

[1]Sophomore, Undecided EECS
[2]Senior, Aerospace Engineering
[3]BS, Mechanical Engineering
[4]BS, Electrical Engineering, Graduate Student, Temple University

Several ideas were put forth, mainly differing in the placement and automating mechanism behind the collector mirror. This included a motorized mirror mounted on a horizontal rail system, a rotating horizontal mast, and a stationary ring of collector mirrors mounted at the base of the mast. However, due to time constraints, more simple designs were chosen.

A. Mechanical Design

Three prototype designs were developed as test beds for the solar-redirecting-tower concept. Designed with speed and ease of construction in mind, the later designs were envisioned as upgrades of previous models to ensure that the research team could begin testing as soon as possible.

Mark I

Illustrated below in Figure 1 the Mk. I prototype was designed as a simple, bare-bones design that could be easily constructed with existing materials in conjunction with those that could be cheaply and quickly acquired.

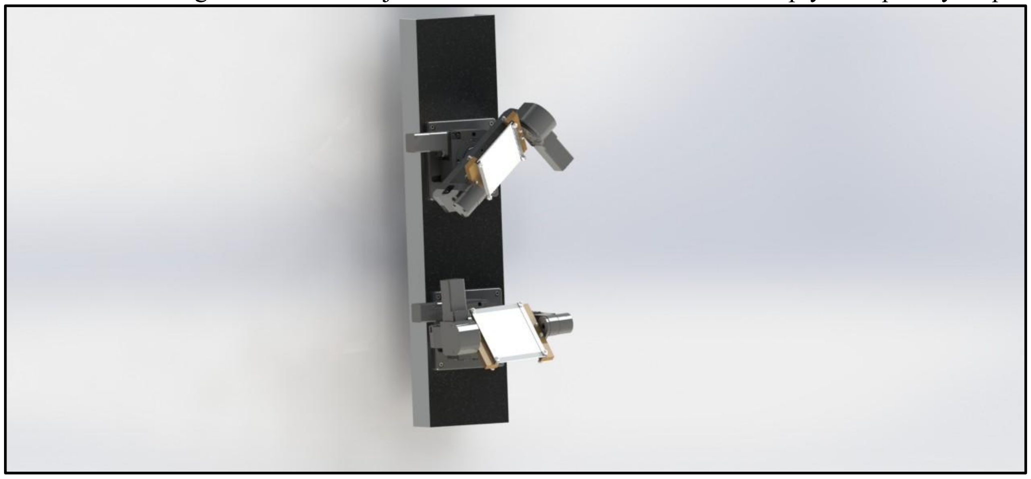


Figure 1. CAD Drawing of Mk. I Prototype

Consisting of two mirror-gimbal assemblies mounted to the side of a vertical support member, Mk. I offered the opportunity of a cheap, simple, test bed for testing our original proof of concept as well as software and sensor interfaces.

Mark II

The Mk. II prototype was an attempt to improve upon the Mk. I prototype, mainly increasing the field of solar transmission

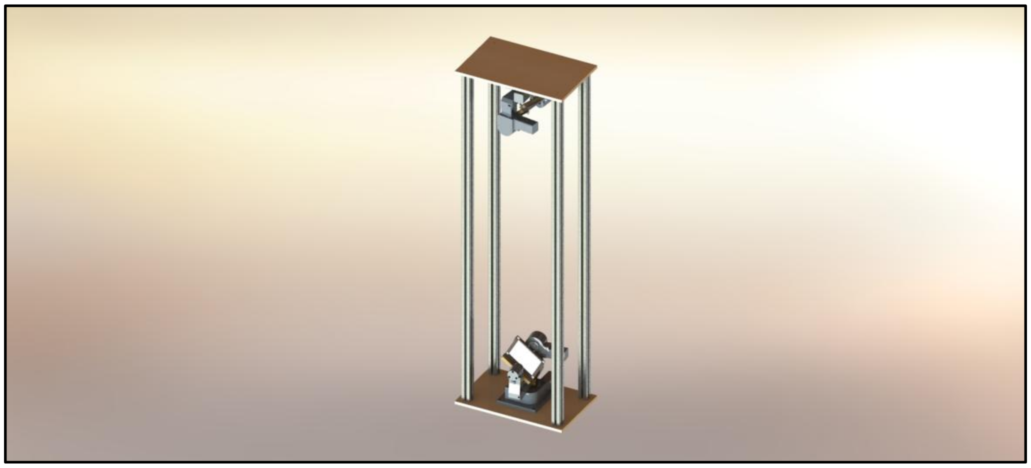


Figure 2. CAD Drawing of Mark II Prototype

As seen above in Figure 2, the Mk. II test article prototype maintains most of the simplicity found in the Mk. I, while improving upon its flaws. The gimbals are mounted horizontally to simplify the algorithms developed to control the system. The top relay gimbal, mounted upside down in the configuration shown above, can redirect light rays in a full 360° even though the gimbal only has ±90° of motion in each of the two degrees-of-freedom. The rest of the frame consists of two plywood sheets and four 5-ft sections of 80/20 aluminum extrusions, all of which are easily acquired from existing materials. Due to the simplicity of the prototype, both from mechanical and software standpoints, the team elected to forgo construction of Mk. I and proceed with development of the Mk. II prototype.

Mark III

The Mk. III prototype is designed as an upgrade of Mk. II, designed in an attempt to optimize the reflected optical train of sunlight hitting the mirror apparatus at low incidence angles around the horizon. Shown below in Figure 3, an "umbrella" ring of mirrors mounted on the top of the Mk. II frame would form a tertiary set of mirrors to reduce the angles of reflection throughout the optical train. This would maintain the maximum surface area hit by reflected light, theoretically increasing the amount of light reflected to a target. Initial simulations were developed to compare the efficiency of two-mirror and three-mirror systems and while three-mirror systems showed higher efficiency in low-incidence-angle light, the increase was not necessarily enough to warrant the increased complication and weight of the tertiary mirrors.

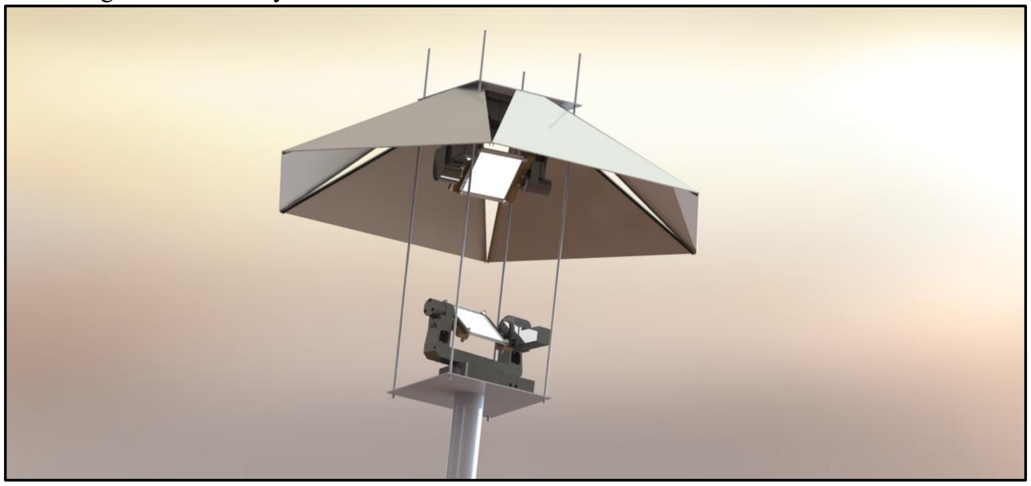


Figure 3. CAD Drawing of Mark III Prototype

B. Electronic Software/Hardware Design

1. *Preliminary Design*

A variety of different options for use in the electrical and software sides of the project were explored. Exploration began by considering an approach that would use FIRST Robotics libraries with a proprietary FRC cRIO image to control the system. This implementation would allow for the use of Java directly loaded onto the cRIO for easy module input, system control, and a simpler experience all around. However, due to the lack of software required to use, the lack of specialized FRC hardware required, and the wrong type of cRIO, this approach was abandoned.

Next, an approach using solely LabView and some C to interface with the devices attached to the cRIO was briefly considered. This was quickly abandoned because LabView programming environment creates confusing control structures that cannot be easily followed by someone trained in traditional software engineering.

After this approach was abandoned, another method was discovered. Implementation of dynamic linked libraries, or shared libraries; to talk to the gimbals using their specific pre-implemented commands was initiated. Methods to control the gimbals using a standalone CLI program for testing purposes were created. The DLL was then imported into Java using JNI for use in higher-level control code. The aim of this was to simply use C as a kind of wrapper for the gimbals to allow the higher-level Java code to control them with more logic and purpose.

A GUI was created for the gimbals to be controlled via keyboard and has been successfully tested. The program will eventually be able to control the gimbals using a Joystick or Xbox 360 controller as well as display vital gimbal information. Upon attempting to move the code to the Windows Embedded 7 environment on the cRIO, no problems were encountered. Successful gimbal control using the keyboard from the cRIO and gimbal speed adjustment with further keyboard input was achieved. However, later on when trying to use the modules that plug into the cRIO to collect data from sensors (initially just using a potentiometer to see if we can get analog voltage data), it was discovered through a long day of trying to install a variety of drivers, that LabView is the only way to talk to the modules.

After several phone calls with National Instruments, it was determined that LabView would have to be used to for the modules that could then be used in a wrapped class written in C, that could finally be used in Java using JNI. This approach is currently under development and testing and will be further implemented upon completion of such testing.

2. *Electronic Hardware Overview*

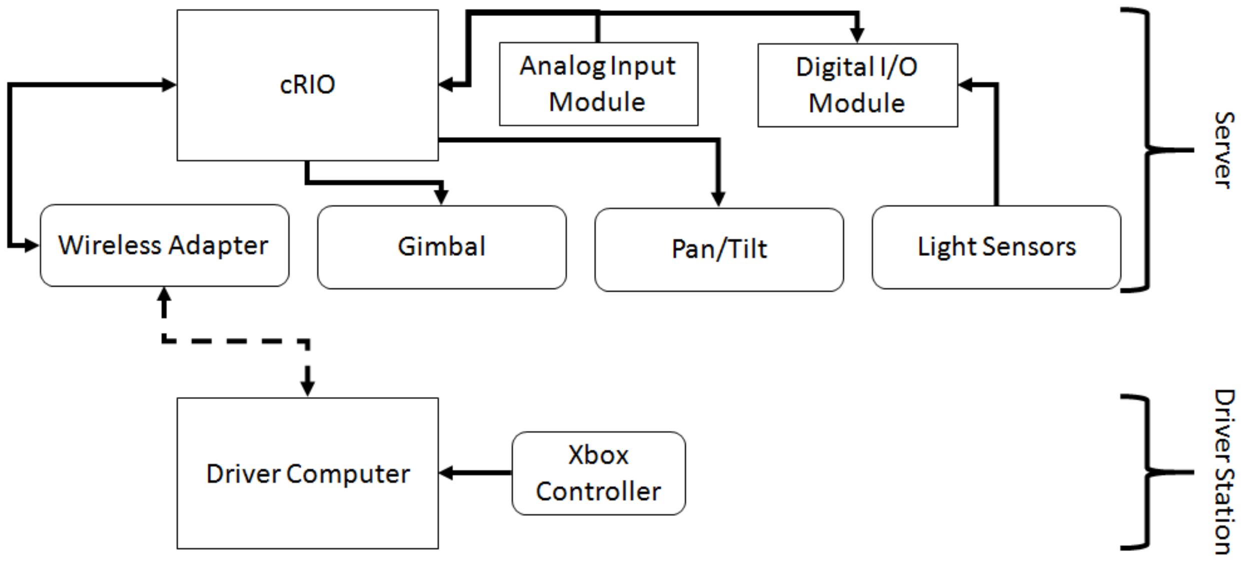
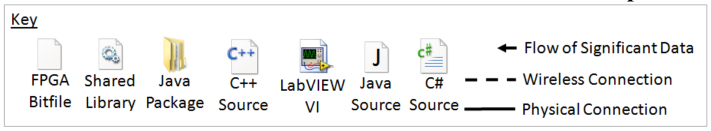


Figure 4. Block diagram of the electronic and electromechanical hardware comprised in SOLARBEAM



As shown in the figure above, the electronic hardware of the SOLARBEAM system can be broken down into two subsystems, the driver station and the server. Through the driver station subsystem's remote computer, the user can establish a network connection and control the server's cRIO Embedded Multicore Computer System, along with the other electronic/electromechanical hardware connected to the cRIO. The driver station subsystem's major components simply consist of a Remote Computer and an Xbox Controller. The server subsystem consists of the cRIO, RIO Digital I/O and Analog Modules, Gimbal's Stepper Motor and Controller, Wireless Adapter, and the Pan/Tilt Gimbal's Servomotors and Controller.

CompactRIO (cRIO) Embedded Multicore Computer System

The CompactRIO 9081 Embedded Multicore Computer System, referred to as cRIO in this document, is a "high-performance multicore system" developed by National Instruments for embedded applications requiring deterministic timing behavior. In the context of the project, the cRIO's capabilities are employed to run software and control the hardware, while taking advantage of certain features of the cRIO such as its FPGA and Windows Embedded Standard 7 Operating System (OS). The Analog Input Module was intended to connect to the Ultraviolet Sensors, which was dropped since the Light Sensors operated with higher performance. The Digital I/O Module connects to the 3 Light Sensors. USB connections connect to the cRIO's Wireless Network Interface Adapter, Gimbal Stepper Motor Controller, and Pololu Servomotor Controller. The Ethernet connection is normally used to connect to the cRIO with a Remote Computer, typically for debugging purposes and/or loading updated software to the cRIO. Though, during the initial stages in the SOLARBEAM system's development, a USB keyboard, USB mouse, and VGA monitor were connected to the cRIO for the same purposes. The cRIO's documentation reports there are two available options on what Operating Systems (OS) the cRIO can support, which are either LabVIEW

Real Time or Windows Embedded Standard 7 (WES7). For reasons explained in this document's software overview section, the selected OS is WES7.

.

Digital I/O Module

The NI 9403 RIO Digital I/O Module is a module designed by National Instruments and attachable to the cRIO's 8-slotted chassis for general digital input/output purposes. In the context of SOLARBEAM, the Digital I/O Module's last 24 of 32 pins, i.e. pins 31-8, are programmed to solely act as general purpose input and output (GPIO) pins. The GPIO pins, though, are not intended to function with precise timing since the software is developed such that the outputs of the pins are only changeable within WES7, which is not a Real Time Operating System (RTOS). Thus, only the 3 Light Sensors are connected to the GPIO pins since the Light Sensors utilize a communication protocol (i.e. I2C) not reliant on precise timing. Other devices that meet the Digital I/O's specifications and do not require specific timing constraints are compatible with the Digital I/O. The remaining pins, i.e. pins 7-0, are programmed as reserved pins. The first 4 of the reserved pins were intended to drive PWM signals to control the servomotors; however, the PWM pins were later abandoned for reasons described in this document's software section.

Analog Input Module

The NI 9201 RIO Analog Input Module is simply an 8-channel analog-to-digital converter (ADC). Although software was developed for the Analog Input Module, the module serves no function in SOLARBEAM. The Analog Input Module was intended to convert the analog voltage-outputs of the Ultraviolent Sensors to digital information comprehendible to the cRIO's software. Because the Ultraviolent Sensors are now entirely replaced by the Light Sensors, the Analog Input Module is only kept in case its functions become necessary later in SOLARBEAM's development.

Gimbal's Stepper Motors and Controller

The gimbal only referred to as the Gimal (not the Pan/Tilt) is composed of the GM6 Gimbal and the NSC-A2L Two Axis USB Stepper Motor Controller, both of which are manufactured Newmark System Inc. In SOLARBEAM, the Gimbal's purpose is to orientate the relay mirror and its relayed solar beam in a direction chosen by the user. In the middle of the SOLARBEAM system's development, there once was a second GM6 intended as the carrier of the collector mirror. Because of an accidental failure that resulted in the GM6's x-axis being damaged, the GM6 was removed from the project.

Pan/Tilt Gimbal's Servos and Controller

Unlike the Gimbal, the Pan/Tilt's gimbal is controlled by two servomotors and a sevmotor controller. The motors are both HS-785HB servos developed by Hitec and the controller is the Micro Maestro 6-Channel USB Servo Controller developed by Pololu Robotics and Electronics. The servo controller is simply referred to as the Pololu Controller in this document. The Pan/Tilt is the carrier of the collector mirror. Thus, its primary role in SOLARBEAM is to rotate along its azimuth axis such that the mounted collector mirror always reflects the most light as possible to the relay mirror. The Pan/Tilt's servos are only positioned within a range of 360 degrees since multiple rotations only added unnecessary complications to the software's development. And, as mentioned, the damaged GM6 gimbal was removed from SOLARBEAM. The Pan/Tilt is the replacement.

Light Sensor

The 3 Light Sensors selected for SOLARBEAM are all BH1750 Light Intensity Sensor Breakout Boards developed by DFRobot. The BH1750 is an Integrated Circuit (IC) developed by Rohm Semiconductor for applications that require the need to determine the intensity of ambient light. In order to optimize the performance of the autonomous tracking algorithm, the 3 Light Sensors are not connected over the same I2C bus, which is possible due to the nature of the I2C protocol. Instead, each Light Sensor is connected to the Digital I/O Module's GPIO pins over their own respective I2C bus so that the cRIO can simultaneously communicate to all three Light Sensors and thereby avoid having to delay before every Light Sensor's reading. The Light Sensors each require at least 120 milliseconds of delay before each the high-resolution intensity can be read.

Wireless Adapter

The Wireless N150 Pico USB Adapter, manufactured by D-Link, provides a host device connected to its USB interface access to wireless local area networks. While connected to the cRIO's USB, the Wireless Adapter's main purpose is, of course, to enable the driver station to remotely connect to the cRIO without having the need to physically connect with an Ethernet cable. Though, a physical point-to-point connection can be established for certain situations; for instance, there might be a need for a physical connection's slightly better performance or the wireless network established by the cRIO on startup does not operate, correctly. Through either a physical or wireless connection with the cRIO, the user is able to access, modify, or even execute the cRIO's contents through tools such as File Sharing or Microsoft's Remote Desktop Connection. (Wireless Adapter IP Address: 169.254.101.15 ; Wired Ethernet Adapter IP Address: 169.254.119.1)

Driver Computer

There is no specific computer necessary to connect to the cRIO and initiate the SOLARBEAM system. Indeed, any computer should be acceptable, as long as the selected computer has the correct java executable (i.e. "SolarDriverStaton.jar"), can run the Java Virtual Machine (JVM), has installed Xbox Controller related files such as the drivers and libraries, and can communicate over a network with the cRIO. In addition to running SOLARBEAM, the Driver Computer's purpose also includes to free the user from having to set up a monitor, keyboard, and mouse with the cRIO just to modify the cRIO's software and other configurations. Moreover, a user whose Driver Computer is wirelessly connected to the cRIO's SolarBeam network will be able to observe the SOLARBEAM system at any angle while still having access to controls or observing messages from the cRIO's software.

Xbox Controller

Microsoft's USB Xbox 360 controller provides a user intuitive control over any application that can benefit from the Xbox controller's intuitive controls and overall familiarity. Specifically, the Xbox controller is the primary method for controlling the SOLARBEAM system, allowing a user to manually control how the relay and collector mirrors are orientated and toggle the system's autonomous light-tracking mode. If the Xbox controller is, for whatever reason, not available, the user does have the option to control SOLARBEAM with the Driver Computer's keyboard.

3. *Software Overview*

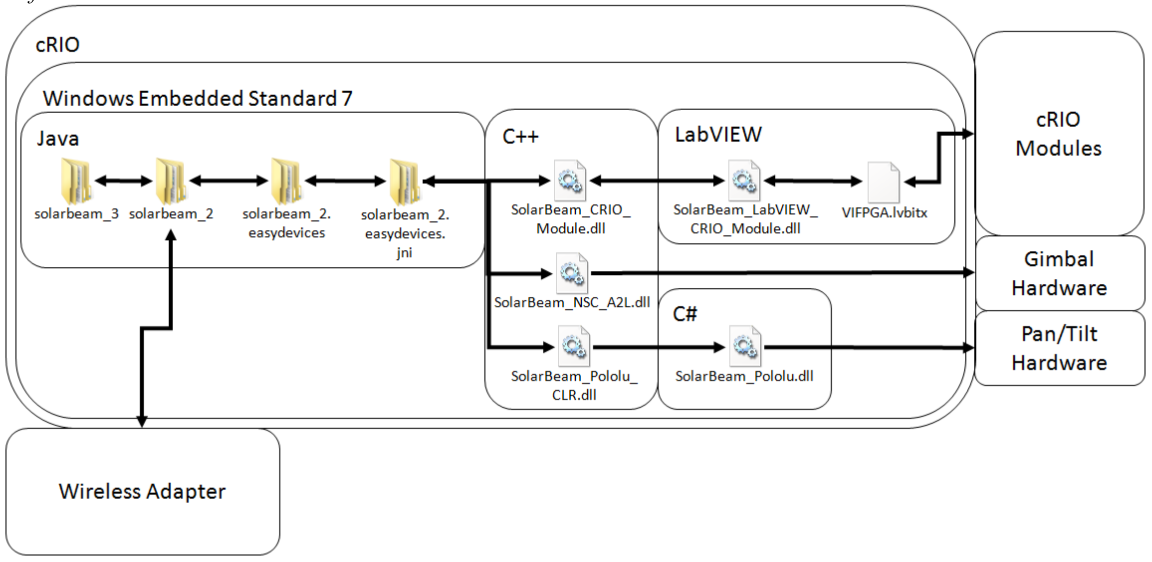


Figure 5. SOLARBEAM's Server block diagram

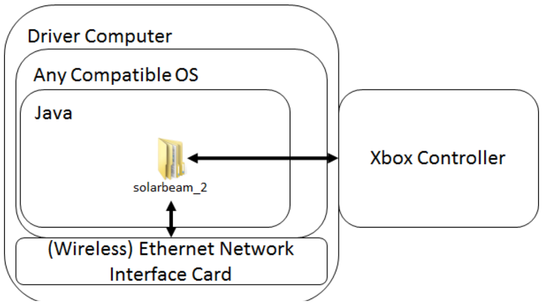
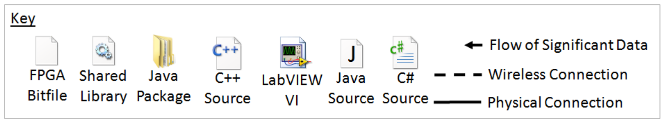


Figure 6. Driver Station's block diagram



The figure above graphically describes the major components of the SOLARBEAM system's software. All of the software runs on either the cRIO's Windows Embedded Standard 7 OS (WES7) or the remote computer's OS. WES7, a smaller version of Windows 7 developed by Microsoft for embedded applications rather than general-use, is employed for its visual similarity with the regular Windows 7 OS and compatibility with software that can run on Windows 7, such as the shared libraries and the drivers required for using the Gimbal Driver. Though, WES7's compatibility is limited. The other option, LabVIEW Real Time OS, has the advantage of being a Real Time OS (RTOS), and as such, has deterministic timing capabilities; however, the time necessary to learn how to program within the RTOS would have reduced the amount of available time in SOLARBEAM's software development. The Remote Computer can be any system capable of connecting to wired or wireless computer networks and running the required software. Thus, the Remote Computer does not have a specific OS. Though, a computer running Windows 7 is preferred for its Remote Desktop Connection (RDC) client-application. RDC allows a user from a computer, the client, to remotely log onto a separate computer, the server, over a network connection and utilize the server's graphical user interface (GUI), including the server's desktop and applications. Along with File Sharing, RDC is the primary method for transferring and running software on the cRIO.

The software developed for SOLARBEAM is mostly written in the programming languages Java, C++, and LabVIEW. The integrated development environments (IDE) in which the software is developed are Netbeans for Java and Visual Studios 2010 for the C++. LabVIEW itself acts as its own IDE. Although software developed in three different programming languages may appear confusing, the combination of the three languages, in practice, is simpler than the alternative option of only using LabVIEW and allows for the advantages particular to each language. As discussed, all programming that has to do with the cRIO's modules (e.g. Digital I/O) must be done through LabVIEW, and learning how to take full advantage of LabVIEW would have taken longer than developing in familiar languages, Java and C++. In embedded applications, Java generally does not operate with the same level of performance as LabVIEW FPGA, however Java offers more familiar resources compared to the other two languages, such as libraries for easily constructing graphical user interfaces (GUIs) and transmitting/receiving information via stream sockets. C++ is used primarily for the development of shared libraries (i.e. dynamic link libraries or DLLs) that act as bridges between the shared libraries created from LabVIEW and the software developed in Java. LabVIEW, as mentioned, is required just to utilize the cRIO Modules, but is supposed to enable more deterministic timing behavior if the programming is done correctly. It is also worth mentioning there is a (very) small shared library developed in C#.

SOLARBEAM's software can be organized into high-level software developed in Java and low-level software developed in the other programming languages. In the context of this document, high-level software refers to the software developed to implement SOLARBEAM's functionality, such as the autonomous tracking mode, manually control, and establish network connection. The low-level software, on which the high-level depends, refers to software developed as mostly shared libraries and provides an interface between the high-level software and device drivers/other shared libraries needed to access the cRIO's electronic hardware. As seen in the software block diagrams, the high-level software starts to the left and then transitions to low-level software towards the right.

The rest of this document's software section is organized with concise descriptions describing particular sections of the software, starting with the high-level software and ending with the low-level software written in LabVIEW's FPGA.

Java Packages

Different Java classes were created in order to facilitate easy control of the tower using the JNI abstraction layer to call the functions to directly speak with the hardware in C++. These classes worked together in order to create an easy, flowing control package. The following is a brief description of all of the Java classes written to control the tower.

Commands is an interface for DriverStationKeyboardSupport as well as SolarbeamServer. It allows for the two classes to have common constants so that when the constants are sent over the Communication network protocol, they can be understood by both classes.

ControlContainer implements functions for both the Xbox controller and keyboard. The class is structured so that the user instantiates either a Xbox Controller object or a keyboard object to the base type Controller. This approach allows another developer to easily switch between the two controllers and call the same functions in the software's main program.

DriverStationKeyboardSupport is a class that runs locally on any PC that can get on the wireless network broadcast from the cRIO. It sends commands to SolarbeamServer over the Communication protocol that are based on the states of different inputs from either the Xbox controller or the keyboard.

PanTilt is a class that represents the bottom pan/tilt gimbal. It has two HitecServoMotors called pan and tilt, and a great deal of different functions that allowed for the control of the two servos.

PanTilt_Arduino is a class that was used to control the two servos using an Arduino as the PWM controller. This class is depreciated as we are now using a Poulu servo control board to control the servos. The Arduino was not able to be used as there were issues installing its drivers on the cRIO.

SolarbeamServer is the class that runs on the cRIO at startup, waits for a connection from the Driver Station, and then handles the majority of the control of the tower as a whole. This is also where the autonomous tracking code is implemented.

EasyGimbal is the class used to extend the JNI Gimbal class to provide easier functionality. It allows for a secondary later of abstraction from the JNI class which provides for easier object construction and control.

Gimbal is the JNI class that allows for Java to call the native C++ functions that send different commands to the Gimbal's driver box via USB. This allows for full control of the Gimbal.

CRIOModule is the JNI class that allows for raw input and output from the cRIO's digital and analog modules. It has both digital read and write methods to which you can specify a certain pin, as well as an analog read for getting raw data from different analog sensors.

HitecServoMotor is the JNI class that allows for direct control of the HiTec Servos that are used to control the pan/tilt gimbal. The servos are able to set to specific degrees or microseconds as well as enable or disable themselves.

LightSensor is the JNI class that allows for direct reading of the light sensors. The light sensors communicate over the I2C digital protocol, so pins must be specified for the various required inputs and outputs of the sensor itself.

XboxController is the class that allows for input from the Xbox controller using the JInput library. It extends the library's functionality into a single class for easy controller use. When using the Xbox controller class, every time input is to be checked, the controller must be polled to update information states.

Communication is the protocol developed to allow for the sending of data over network from the DriverStationKeyboardSupport to SolarbeamServer for the wireless control of the Solarbeam tower.

In addition to all of these classes, a recursive light tracking algorithm was created for autonomous tracking of the simulated sun. As mentioned above, it was implemented in SolarbeamServer and could be enabled and disabled using the Xbox controller. The light tracking algorithm works by reading values from each of the light sensors. One to represent the intensity towards the left of the pan/tilt, one for the right, and one for the ambient light. The ambient light value is subtracted from the other values to give data that truly represents the intensity of the simulated sun directed at the pan/tilt.

An offset factor is then created by subtracting the right value from the left and dividing by a fixed constant. Depending on the sign and magnitude of this factor, it can be determined which direction the pan/tilt needs to move as well as how much it should move by. If the offset factor is within set bounds, it can be said that the pan/tilt is 'on-target'.

If the factor is not within set bounds, depending on the sign and magnitude, the pan/tilt moves towards the light source. The algorithm then recurses to get on target until the values are within acceptable 'on-target' bounds.

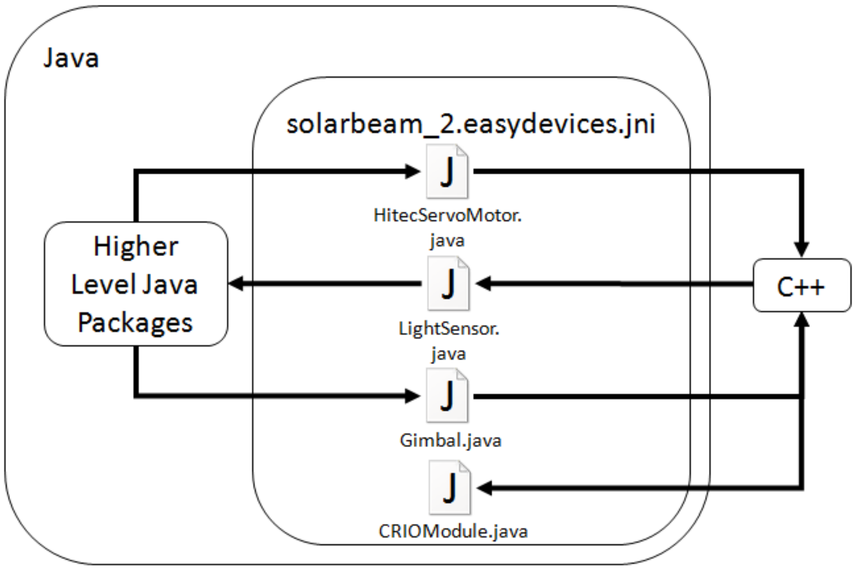
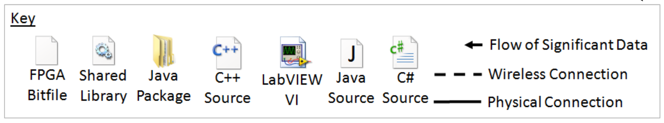


Figure 7. Java classes that utilize the Java Native Interface (JNI)



As shown in the block diagram above, the Java package that contains the low-level Java classes is solarbeam_2.easydevices.jni. All classes contained in the solarbeam_2.easydevices.jni implement the Java Native Interface (JNI), which is necessary to call functions located in the shared libraries developed in C++. Regarding the development of the HitecServoMotor class, an Arduino UNO with software developed to take advantage of the Arduino Servo libraries was introduced after realizing the software developed to implement PWM through the cRIO's FPGA did not provide timing precise enough for controlling the servomotors. The software written for the Arduino-approach was originally tested on a separate computer. Unfortunately, the drivers necessary to interact with the Arduino UNO were not compatible with the cRIO and thus another approach became necessary. Fortunately, the Pololu Controller was discovered as a compatible alternative.

Shared Libraries

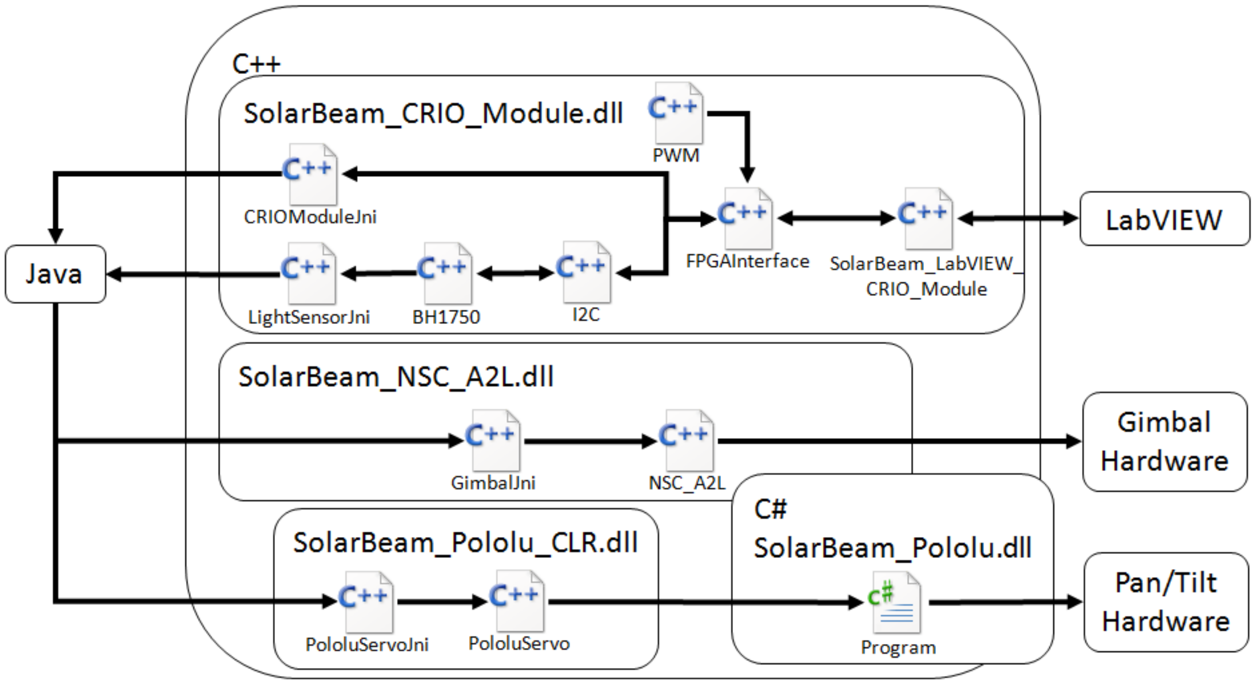
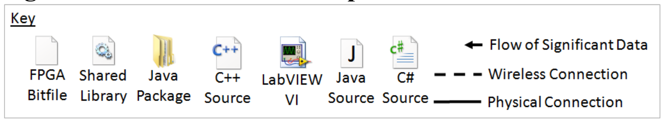


Figure 8. Shared libraries compiled from C++ source files



The primary purpose of the shared libraries is to act as a bridge between lower-level software, such as drivers and other shared libraries, to the high-level software all written in Java. Another purpose of the shared libraries is to implement classes that would hide the details of communicating to a particular piece of hardware from the high-level software. Having to worry less or not at all about the details specific to the hardware allows the high-level software developer to focus all efforts on programming the system's intended function.

The shared libraries on which the higher-level software directly depends are SolarBeam_CRIO_Module, SolarBeam_NSC_A2L, and SolarBeam_Pololu_CLR. SolarBeam_Pololu is a shared library developed differently from the first three and will be discussed with the third shared library. Take note the prefix "SolarBeam_" will be implied whenever a shared library's name is mentioned for the remainder of this document. Also note the source files shown in the block diagram and whose names end with "Jni" have different, much longer names in the actual source code.

CRIO_Module is compiled from all the source files that depend on the LabVIEW_CRIO_Module and implement the Light Sensor, I2C, PWM, and JNI functionality. The **NSC_A2L** shared library is developed such that the high-level software developer would have simple means of accessing and controlling the NSC-A2L 2-axis Stepper Motor Controller in Java. Because the shared library operates through USB instead of the cRIO's FPGA, the shared library is compiled on its own. The shared library includes the following source files The **Pololu_CLR** and the **Pololu** shared libraries, which are collectively referred to as the Pololu libraries, are developed to allow the high-level software developer to operate the Pololu Controller with intuitive functions in a Java application. The reason for compiling the Pololu shared libraries separately from the first two shared libraries is the fact Pololu depends on the Common Language Runtime (CLR) virtual machine (VM) and appears to diminish performance when included with CRIO_Module shared library. Unlike every other shared library developed for SOLARBEAM, Pololu is actually a library assembly developed in C# that refers to library references developed by Pololu Robotics and Technology.

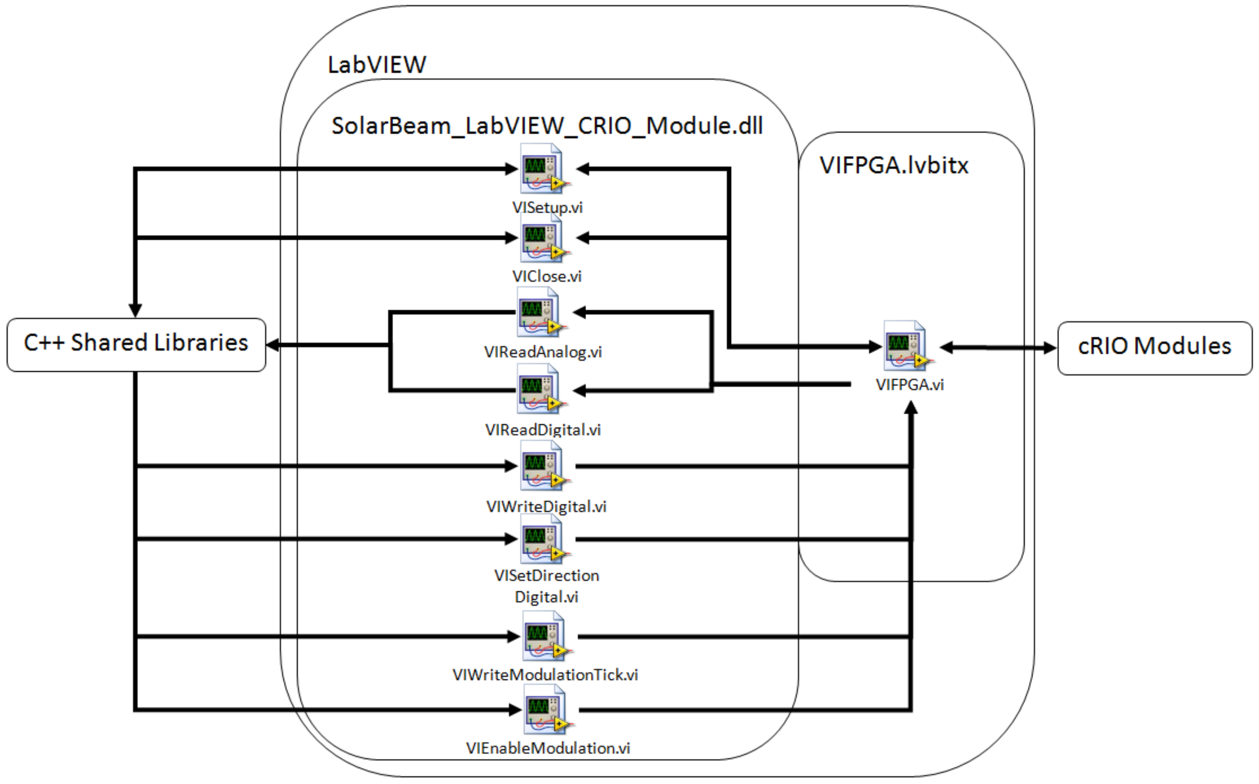
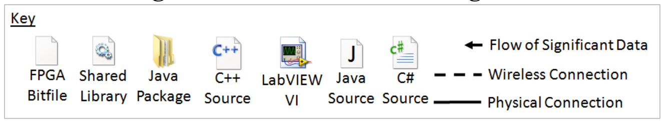


Figure 9. LabVIEW block diagram



The **LabVIEW_CRIO_Module** shared library and the VIFPGA bit-stream on which the shared library depends are developed so that higher-level software can access the cRIO's Analog Read and Digital I/O Modules. Both the shared library and the bit-stream are compiled from VIs created in LabVIEW. The VIs are created with the idea of limiting programming done in LabVIEW down to general operations so that the tasks performed over the FPGA and its connected RIO Modules can be extended. Thus, less time is spent trying to implement complex algorithms in a less familiar language and the software becomes more organized since certain complexities are localized in only certain areas of the software. For instance, high-level tasks such as autonomy and user control are done in the Java, communication protocols and particular details specific to hardware is done in C++, and very general access to hardware is done in LabVIEW. General functions the VIs perform include connecting/closing connection with the FPGA, reading analog information from the Analog Input Module' 8 channels, reading or writing digital information to or from the Digital I/O Module, and other functions.

III. Testing

Upon construction, the Mk. II prototype was transferred to the Flight Robotics Laboratory for installation. Mounted to the top of an 8-ft tower the test article was oriented towards a bank of ARRI Daylight spotlights, one of which was used as a sun analogue. Initially only the top relaying mirror was used to redirect light in order to optimize the gimbal control algorithms. Once this was completed, the lower collecting gimbal was manually oriented towards the source light, the mirror locked to redirect light up to the secondary mirror. The system functioned as designed, the relay mirror able to redirect light at any location in the Flight Robotics Lab, including those not illuminated by the spotlight. Thusly, the research team completed its first objective; prove the concept of redirecting solar energy to a target in shadow.

Following this milestone, the team began working to upgrade the reflective material mounted to the test article with the hope to further optimize the optical train. The original material, a reflective aluminum sheet metal, proved to be quite reflective but its thinness meant that it was prone to warping during installation. These small defects would cause scattering of the reflected light which would negatively affect the final output of solar energy. Two candidates were chosen: a stretched Dura-lar film and mirrored polycarbonate. Similar 5x5 inch squares of each material were cut and mounted to the test article. All three configurations were illuminated by our sun analogue at the same angle of incidence, with the relaying gimbal targeting a stationary solar panel in a shadowed portion of the Flight Robotics Lab. A multimeter was used to measure the ambient voltage, as well as the voltages given off upon exposure to each setup's optical train. The results of these tests are shown below in Figure 4.

Figure 10. Results of Reflective Material Testing

Material	Single Mirror Voltage	Double Mirror Voltage
Dura-lar covered sheet metal	6.7V	5.6V
Reflective sheet metal	7.5V	6.4V

IV. Conclusion

Over the course of ten weeks, the research team has managed to develop a proof of concept into a functioning test article that can redirect sunlight to shadowed areas. Two of the group's three objectives were met, proving the solar power transmission concept and developing autonomous sun-tracking algorithms.

For future work, the research group proposes research be done into specialized mirrors, focusing lenses, or beam collimators to increase the efficiency of transmitting sunlight and the development of auto-tracking capability for a target rover platform.

Appendix A. Personal Contributions

Luc Bettaieb

The project to which my team and myself were assigned to was to come up with a proof-of-concept prototype system to redirect artificial sunlight generated by 6kW ARRI Daylight spotlights at a specified target in the flight robotics laboratory. After some brainstorming, we came up with the design of a vertical array of two gimbals fitted with mirrors that would be mounted on top of a tower in the flight robotics laboratory.

These gimbals would be controlled using their respective driver boxes connected to a National Instruments Compact RIO (cRIO) 9081. The software and electrical team (comprised of Andrew Powell and myself) initially attempted to use the FIRST Robotics control system configuration for control and automation of the system, but due to hardware differences and restrictions, this proved to be impossible. We instead adopted a different control configuration. On the cRIO, we installed Windows Embedded 7 so that we could run high-level Java control software to interface with low-level software that would directly control the hardware.

To keep the all of the code organized, I set up a Google code subversion repository. This would allow us to keep track of any changes made in the software over the course of the summer and have it synced across all of our computers. In addition, it also provided an online backup of all of the software written in case of any kind of system failure.

I was in charge of writing the high-level Java control code that would run on the cRIO. The software was able to interface with the hardware through Dynamic Linked Libraries (DLLs) written by Andrew. These libraries, written in C++, were implemented in Java using a programming framework called Java Native Interface (JNI) that essentially was able to allow Java to call the low-level functions associated with the DLLs. The high-level code that I wrote was initially going to be comprised of various classes used to encapsulate the different sensors, aggregate the data collected by them, and use that to control the gimbals. There was a main SolarBeam class that would run on the device and handle all of the main logic flow, a SolarSyncer to keep the mirrors in sync, a SunTracker to provide a translational vector for the collecting gimbal to keep it pointed at the simulated sun, and an XboxController class so that an Xbox controller could be used to manually control the system. However, due to one of the gimbals breaking, the way that the wireless communication protocol would work, and time constraints, this approach was abandoned for a simpler one.

Construction of the tower was primarily done using the machine shop in 4487 where we worked for the majority of the summer. I assisted in construction using my machining skills developed over the years I spent participating in FIRST Robotics in high school, as well as formal instruction during the second semester of my freshman year at Case Western Reserve University. We drilled, tapped, and reamed holes in the aluminum 80/20 supports for the tower, attached those supports to panels, and mounted the gimbals on those panels.

After the tower was constructed and moved to the flight robotics lab, more software implementation and testing could begin. A servo pan/tilt mechanism was found and mounted as a replacement for the broken gimbal. Control software and DLLs were developed so that the new pan/tilt mechanism could be controlled with the Xbox controller. In addition to the other changes, a new software schema was developed. The new schema was implemented to make better use of the communication protocol and simplified tower that was now reality. A DriverStation class was written that would run on a host laptop to control the tower remotely. This was achieved by sending bytes that correspond to different commands over a secure wireless network. A new Solarbeam class was also written that would receive the commands sent by the DriverStation and perform the corresponding actions.

A tracking algorithm was also developed. The idea behind the algorithm is to use two light sensors with horizontal limiters around them to cause the majority of the light sensed to come from the horizontal plane. The algorithm receives data from the two sensors, and creates an offset factor based on them. Ideally, if the collecting pan/tilt is on target, both sensors should have the same reading. So if the difference between the two collected values is zero (within some tolerance), it can be said that the apparatus is on target. The offset factor is checked against the base case, and if it on target, the method ends. Depending on how far the factor is from the base case, the values are adjusted by panning the collector and the method is recursed. Eventually, the values become within an acceptable ‘on-target’ tolerance. This algorithm is looped until disabled so tracking will always be achieved.

In addition to software, I took on the responsibility for creating the academic poster and editing its content. Having previous experience creating academic posters in Adobe InDesign, I used that program and created a template similar to the one used by the previous Robotics Academy’s FeatherSail team. The end result was a simple, crisp looking, and effective poster to be presented at the poster session.

Christopher Goodeaux

My responsibility during the initial design period was to devise concepts for a test stand to prove the solar-redirecting-tower concept, namely the solar farm-style collecting mirror and the over-under configuration used on the Mark II test stand. The goal of the ring of solar panels was to maximize the area of the collecting mirror’s reflective surface so that it could always transmit some amount of light to the relaying mirror regardless of the sun’s orientation. The ring also had an advantage of simplicity when compared again other collector mirror designs, most of which involved series of motors and moving parts. The simple over-under design incorporated into the Mark II test stand was developed in an attempt to further maximize simplicity in order to quickly develop a proof-of-concept for redirecting sunlight.

Next a basic CAD drawing of the Newmark Systems GM-6 gimbal, procured for use on the project, was created for insertion into our design assembly. A bracket system was also designed to hold each mirror to a gimbal firmly, but also to promote ease of access in order to test different reflective materials. The completed CAD model was used to find and order parts for construction.

Finally research was performed on the lighting conditions of the Lunar South Pole, specifically large impact craters like Shackleton and Shoemaker, in order to factor them into our design constraints. It was discovered that locations on the rim of the impact craters see almost continuous sunlight throughout the year at incidence angles very close to the horizon.

Construction and mounting of the Mark II test article was completed primarily by Vadim Linevich and myself. Holes were drilled in the plywood bases to attach the gimbals and supporting aluminum extrusion columns, and the test stand was constructed using the appropriate fasteners. The completed article was transferred the Flight Robotics Laboratory and mounted on an 8-ft tall tower across from a bank of 60kw spotlights.

Using the bank of spotlights as a sun analogue, testing was initiated in order to prove our concept of redirecting low angle-of-incidence sunlight. Testing confirmed that our article could indeed beam light into darkened areas of the lab. After this was completed, the reflective sheet metal used for the initial construction was replaced with different materials in order to compare their reflective properties. Lastly, a pair of light sensors were installed onto the bottom collecting gimbal for testing suntracking software developed my Luc Bettaieb and Andrew Powell.

Vadim Linevich

Overall, I worked closely with Chris Goodeaux on mechanical aspects of the Solarbeam system. Together, we took the brainstorming ideas and created designs of three different prototypes. In addition, I served as a team leader. As a team leader, I scheduled and coordinated tasks, handled paperwork (such as filling out part order forms), and facilitated creative, productive, and safe work environment.

In the beginning of the project, I spent a significant amount of time researching and experimenting with various optical train designs. Success of the project was heavily depended on a choice and configuration of optical components. I proposed to use two mirrors in MK II design, and three mirrors in MK III design. I sketched optical train designs using paper and pencil tools to quickly analyze and validate feasibility of designs. I refined optical train designs in CAD software, namely Solidworks. Having these analysis allowed us to further shape our vision of Mk II and Mk III designs.

I developed a dynamic sketch in Solidworks where a designer could specify locations of two mirrors and the light source. After doing so, designer could rotate the mirrors and watch a ray diagram update instantaneously. Based on the design constraints and information about the test facility, I was able to use this sketch to optimize our optical trail. I also built a similar sketch for an optical design featuring there mirrors.

I took a lead role working with CAD (computer assisted design software). I created CAD models of many individual parts and implemented those parts to generate full assembly models. Chris and I used CAD to go through an iterative design process. I ran a few FEA simulations on Mark II and Mark III prototypes in order to absolutely sure that the prototype would be able to safely withstand expected loading conditions.

I wrote a MATLAB program to theoretically evaluate efficiencies of chosen Mk II and Mk III optical designs. The program calculated percent of light flux when the light beam was reflected by a mirror. Based on this analysis we decided to focus on Mk II prototype.

I strived to minimize cost of the prototype without sacrificing its functionality. Before sending out order forms, I price checked every part with several vendors. I planned ahead and took extra steps in ordering raw materials to minimize waste in part manufacturing process. I also looked for available scrap material and tried implementing available resources in a design.

Chris and I did most of the manufacturing and assembly work. The first prototype was built according to our blueprints. Throughout the project, we had to continuously modify and adjust the system for various reason. For example, we have experienced some hardware failures, so we had to find and retrofit suitable replacements.

I also spent a little bit of time programing an Arduino board to read light sensor data and send signals to servo motors in order to achieve autonomous light tracking. This was a plan B option which was abandoned, since plan A option was successful.

Andrew Powell

Andrew Powell's contributions to the SOLARBEAM project can be summarized as figuring out how to properly connect the project's primary embedded computerized system, National Instrument's cRIO, with other electronic hardware and developing low-level software that would enable the other software developer, Luc Bettaieb, to access the hardware within his high-level software. Regarding the project's electronic hardware, Powell assisted in ensuring the cRIO had the correct power supply selected the appropriate RIO Digital Input/Output Module; figured out how to properly connect the cRIO to certain electronic hardware with the cRIO's swappable RIO modules; and performed other smaller electronic hardware-related tasks, such as helping to wire together the project's final prototype.

Most of Powell's contributions were in the development of the project's low-level software, and a small portion of the high-level. The high-level portion includes the project's Java Communication class that facilitates communication between two machines, which, for SOLARBEAM, are the cRIO and a remote computer. In addition to the Communication class, Powell wrote a series of Java classes, utilizing the Java Native Interface (JNI) in order to call the low-level software developed as shared libraries. Because of the cRIO's requirements and the fact the team chose Java as the primary programming language for the development of the project's high-level software, Powell's low-level software was mostly developed in the programming languages C++ and LabVIEW as shared libraries (also referred to as dynamically linked libraries (DLLs)). The integrated development environments (IDEs) in which Powell developed the low-level software are Microsoft's Visual Studios 2010, for mostly the C++, and National Instrument's LabVIEW.

Powell created the shared libraries SolarBeam_LabVIEW_CRIO_Module.dll, SolarBeam_NSC_A2L.dll, SolarBeam_CRIO_Module.dll, SolarBeam_Pololu_CLR.dll, and SolarBeam_Pololu.dll. In order to generate the first shared library, Powell created a series of LabVIEW virtual instruments (VIs) and compiled a bit-stream that enabled general access to the RIO Analog Input and Digital I/O Modules via the cRIO's Field Programmable Gate Array (FPGA). The VIs created performed functions such as initiating/closing a connection with the FPGA, writing to/reading from the pins on the Digital I/O Module, and reading from the Analog Input Module. For all the shared libraries developed in C++, which includes SolarBeam_CRIO_Module.dll, SolarBeam_NSC_A2L.dll, and SolarBeam_Pololu.dll, tools from JNI were used to make the C++-developed functions callable in Java. Included within SolarBeam_CRIO_Module.dll, Powell implemented the Inter Integrated Circuit (I2C) class, the Light Sensor class, and other C++ classes, all of which rely on the shared library developed in LabVIEW in order to function. For the purpose of controlling the project's stepper motor controller, the SolarBeam_NSC_A2L shared library was created. Powell constructed the shared library to transmit commands to and receive information from the stepper motor controller via a set of lower-level functions defined in shared libraries created by Newmark. The functionality of the implemented commands range from calibration of the motor controller with the stepper motors' positions, to directing the stepper motors to turn to specified positions, to even changing the speed at which stepper motors turn. Finally, Powell developed the SolarBeam_Pololu and SolarBeam_Pololu_CLR libraries so as to control the Pololu Servomotor controller that drives the Hitec HS785HB servomotors on the project's Pan and Tilt gimbal. The first of the two Pololu libraries was developed in the programming language C# in order to take advantage of proprietary C# libraries that can drive the servomotor controller. The second Pololu library acts as a bridge between the Java-developed software and the C# library. The functions Powell programmed into the two Pololu libraries allow a Java-software developer to establish/break a connection with the controller and position a servo via the controller. Prior to writing the Pololu shared libraries, Powell tried to develop software for directly controlling the Pan and Tilt's servomotors by first configuring the cRIO's FPGA to run Pulse Width Modulated (PWM) signals through a few pins of the Digital I/O Module. Powell also implemented an Arduino UNO such that the Arudino would be programmed to receive serial commands via USB and then its Servo libraries would then control the servomotors.

In addition to the technical tasks described in the paragraphs above, Powell also contributed to many of the pictures and writing seen in this document's Electrical and Software Design section.

Acknowledgments

The team would like to extend its gratitude toward Dean Alhorn (ES32) for his help and oversight as the principal investigator, Ricky Howard (ES32) for all of his help and coordinating for the project. Special thanks goes out to Chris Randall, Frank Six, Gerald Karr and Jason Entenmann for their leadership and help in running the 2013 NASA Marshall Space Flight Center Robotics Academy.

References

[1]Bussey, D. B. J., McGovern, J. A., Spudis, P. D., Niesh, C. D., Noda, H., Ishahara, Y., and Sørensen, S. -A., "Illumination conditions of the south pole of the Moon derived using Kaguya topography," *Icarus*, No. 208, 2010, pp. 558-564.