Automation of the Spectral-Line Imaging Camera for the Virginia Tech Spectral-Line Survey

Kenneth P. Portock

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Master of Science
in
Physics

Committee
Dr. Brian Dennison
Dr. John Simonetti
Dr. John Broderick

December 9, 2002
Blacksburg, Virginia

Keywords: Astronomy, ISM, WIM, Observatory, Automation, Hydrogen alpha, CCD, Telescope, PERL, BASIC Stamp

Automation of the Spectral-Line Imaging Camera for the Virginia Tech Spectral-Line Survey

Kenneth P. Portock

(ABSTRACT)

The Virginia Tech Spectral-Line Survey (VTSS) is a high sensitivity, high resolution survey of Hydrogen-α and Sulfur II emission in the full northern hemisphere. The instrument used for the survey is the Spectral Line Imaging Camera (SLIC). SLIC uses a fast(f/1.2) lens attached to a cryogenically cooled, TK 512×512 CCD with 27 μm pixels. The focal length of the lens is 58mm which gives a pixel size of 1.6 arcminutes. The diameter of each field is 10°. A filter wheel ahead of the lens allows for imaging at different wavelengths. Automating the imaging system is desirable and advantageous due to the large scope of the survey. A variety of devices have been developed in order to implement automation of the SLIC observatory. They include an automated focusing mechanism, filter wheel, liquid nitrogen auto fill system, motorized roll-off roof, cloud monitor, and an equatorial mount. A PERL script, called SLICAR (Spectral Line Imaging Camera Automation Routine), was written to control and communicate with the various hardware and software components. The program also implements a user prepared Observing File, and makes decisions based on observing conditions.

Acknowledgments

I would like to thank the following people for their invaluable support and expertise:

Dr. Brian Dennison
Dr. John H. Simonetti
Dr. John J. Broderick
Phillip Nelson
Roger Link
Melvin Shaver
John Miller
Scott Allen
Fred Mahone
Norman Morgan
Rene Goerlich

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 The Interstellar Medium

For many years astronomers have known that the space between the stars in our Galaxy is not empty. In fact it consists of vast clouds of gas and dust permeated by cosmic rays and magnetic fields. The basic composition of the space between the stars has been measured to be 90.8% hydrogen, 9.1% helium, and 0.12% heavier elements by number[5]. In terms of mass this results in 70.4% hydrogen, 28.1% helium, and 1.5% heavier elements. This great sea of gas and dust is known to the astronomical community as the interstellar medium (ISM). It has become clear through observations that the ISM plays an important role in the Galaxy as a whole. Previously thought to be an almost neutral and static component of the Galaxy, it has become evident that the ISM is as dynamic and complex as the stars themselves. Learning the complexities of the ISM is crucial to understanding the dynamics of star formation and the structure of the Milky Way galaxy.

Each component of the interstellar medium–gas, dust, cosmic rays, and magnetic fields–interact with each other, and with stars, to form various features in our galaxy. As stated above, a large component of the ISM is made up of hydrogen gas. Therefore, understanding the role that hydrogen plays will go a long way towards understanding the ISM. The hydrogen component of the ISM is made up of various phases. They are cold molecular gas (H_2), neutral atomic gas (HI), warm ionized gas (HII), and hot ionized gas (also HII). These phases of interstellar gas have individual properties but they are not independent of one another. Each phase interacts with the other phases in complicated ways.

The molecular hydrogen phase is mostly limited to the coldest, densest clouds. Most starlight can not find its way into these clouds to warm them or dissociate the molecules within. Typically H_2 clouds have a temperature of $\sim$15K, and a number density $\sim$200cm^{-3}[1]. Because H_2 regions are cold and dense they are subject to gravitational collapse, and are known to be regions of active star formation. Molecular hydrogen clouds are mostly found in a very

thin layer at the galactic plane. Surprisingly, H_2 has been discovered in exposed, low density areas far outside the galactic disk. Obviously some other mechanism is at work which is not yet fully understood. Direct observations of H_2 are difficult since its UV emissions are readily absorbed by dense clouds. To find H_2 clouds astronomers look for the trace molecule CO, which is characteristic of dense molecular clouds.

There are two phases of neutral atomic hydrogen. The cold component is found at a temperature $\sim$120K and density $\sim$25cm^{-3}[1]. The warm component is $\sim$8000K and less dense than the cold component $\sim$1cm^{-3}[1]. Radio observations of the hyperfine transition 21cm line yields the structure of HI in our galaxy. Maps of HI clouds show intricate shapes that appear as long filaments or extended sheets with small clumps. The cold component of neutral atomic hydrogen is also found near the galactic plane. The warm neutral medium extends far beyond the disk of cold gas.

The warm ionized gas, or warm ionized medium (WIM) is a phase of hydrogen that is higher in temperature, and less dense than neutral hydrogen, and is not as hot or rarefied as hot ionized hydrogen. It reveals itself by emitting photons produced from ion and electron recombination. The WIM is of particular interest to this research so it is discussed in more detail in section 1.2.

Stretching far above and around the entire Galaxy is a "halo" of hot ionized gas called the hot ionized medium (HIM). This phase, rarefied ($\sim 10^{-3}$cm^{-3}) and heated to $\sim 10^6$K[1], is thought to be produced from supernovae explosions and fast stellar winds. Large pockets of hot ionized gas, called superbubbles, are thought to rise and break out of the galactic plane forming chimneys and fountains of gas that cool and rain back down on the Galaxy.

Cosmic rays and magnetic fields play important roles in the structure of the interstellar medium. Cosmic rays are highly energetic charged particles that can ionize parts of the ISM by colliding with interstellar gas. Cosmic rays can also act as a heating mechanism for interstellar gas clouds. Interstellar matter and cosmic rays are actively coupled through interactions with the interstellar magnetic field. The interstellar magnetic field acts directly on charged particles and this in turn can affect neutral particles through collisions. The magnetic field confines cosmic rays and supports gas clouds against gravitational collapse. Gas clouds are connected to their environment and to other gas clouds via the interstellar magnetic field. Magnetic and cosmic ray pressures can generate instabilities that cause the magnetic field to ripple, and can therefore promote gravitational collapse and star formation.

One of the most important aspects of the dynamics of the ISM is the cycle of material from the ISM to stars and back. Cold dense clouds are where stars are born. When H_2 clouds are massive enough, a star can "condense" out of the fuel rich cloud by gravitational collapse. As it converts its fuel into heavier elements through thermonuclear reactions, the newborn star ejects material back into the ISM by powerful stellar winds. Stellar wind is the continuous outflow of gas from stars. Very fast stellar winds are thought to produce enormous superbubbles filled with hot ionized gas[4]. At the end of its life, the star will send some of its material back into the ISM, as fuel for new star formation. If it is massive enough

it can eject material in one massive explosion, called a supernova, and the expanding shock wave can violently disrupt the surrounding medium. The shock wave may push clouds aside and send hot ionized gas into the galactic halo[1]. The shock wave can trigger gas clouds to collapse in other parts of the Galaxy forming new stars, and the cycle continues.

It is apparent that the ISM plays a significant role in the structure of the Galaxy as a whole. The distribution and characteristics of the ISM determine where star formation occurs and the type of stars that form. In turn it is the ISM and stars together that fashion the overall large scale features of our galaxy. Learning the dynamics and properties of the ISM is motivated by the need to understand the structural aspects and evolution of the Milky Way galaxy.

1.2 The Warm Ionized Medium

The warm ionized medium fills about 20% of the interstellar volume[1]. Faint emissions from the WIM arise from the recombination of free electrons with protons. As the electrons cascade down through excited energy levels photons of different wavelengths are released. One type of photon emitted is from the Hα transition. A photon with a wavelength of 6563Åis emitted when an electron falls to the second energy level from the third energy level. From observations, HII can be found everywhere on the sky, not just near the galactic plane[16]. The WIM is an important and interesting component of the ISM, yet less is known about this phase than any other. What mechanisms are involved in ionizing the WIM? At a temperature of $\sim$8000K[1], what is heating the gas? Questions such as these have not yet been completely answered.

It is known that hot O type stars produce UV photons with sufficient energy to ionize the hydrogen surrounding them. Bubbles of ionized hydrogen, called Strömgren Spheres, are indeed found around these hot stars. Since any hydrogen-ionizing photon is rapidly absorbed in a hydrogen cloud there is a sharp boundary between the ionized hydrogen bubble and the surrounding gas. Because of the prompt absorption of ionizing photons one is led to believe that HII would be confined to the region of space surrounding O type stars, which are mostly created in or near the galactic plane. But observations reveal ionized hydrogen in all parts of the sky, not limited to the galactic plane. How does the ionizing radiation from O type stars reach distant hydrogen? Is there an alternative source of energy which ionizes these regions? Widespread HII suggests that our understanding of the WIM is incomplete.

A particular parameter of the warm ionized medium that can help resolve some important questions is the [SII]/Hα intensity ratio. Complementary observations of [SII] emission, at wavelengths 6716Å and 6731Å, can reveal important information regarding the heating of the WIM when compared with Hα data. [SII] is excited by collisions and therefore is independent of the density of ionizing photons. The [SII]/Hα intensity ratio is related to a quantity called the ionization parameter. The ionization parameter is the ratio of the density of ionizing

photons to the electron density. A large ionization parameter corresponds to a small [SII]/Hα intensity ratio. In traditional HII regions [SII]/Hα is small. But away from the galactic plane the [SII]/Hα intensity ratio is higher. What source can account for this difference? A large scale, high sensitivity, high resolution, survey of Hα and [SII] emission is needed in order to confront these questions.

1.3 The Virginia Tech Spectral-Line Survey

The goal of the Virginia Tech Spectral-Line Survey (VTSS) is to provide a high resolution map of Hα and [SII] emission of the entire northern hemisphere sky to the astronomical community. This survey will elucidate the small scale structure of the WIM, as well as provide important information regarding the global structure of the WIM. The VTSS uses the Spectral Line Imaging Camera (SLIC) designed and built for ISM research. SLIC incorporates a fast (f/1.2) lens with a cryogenically cooled, TK 512×512 CCD. Each pixel is 27 μm and, when used with the 58mm focal length lens, yields an image pixel size of 1.6 arcminutes. The diameter of each field is 10°[3]. Mounted ahead of the lens is a filter wheel which allows observations at different wavelengths. The camera is equatorially mounted on a concrete pier inside a small observatory building with a roll-off roof. The observatory is located at Virginia Tech's Miles C. Horton Research center in Giles County, Virginia.

This survey has been partially completed. Approximately 280 fields in Hα, and 134 in [SII] have been observed. Continuum images of each field are also recorded for processing and analysis. These previous observations account for about 43% of the entire spectral line survey which will ultimately consist of 964 total fields (482 Hα fields and 482 [SII] fields). This survey will be extended to cover the entire northern hemisphere sky and will include 202 additional fields in Hα and 348 [SII] fields. When completed the entire survey will be made available via the Internet.

The VTSS will be compared with a complementary survey, called the Wisconsin Hydrogen-alpha Mapper (WHAM), performed by a research team at The University of Wisconsin in Madison. The WHAM survey provides Hα intensities obtained from spectra, and covers the entire northern hemisphere. The instrumentation used for the WHAM survey is well suited for high spectral resolution studies[17]. While the WHAM offers high spectral resolution, it compromises its spatial resolution because it utilizes a 1° beam. VTSS is a good complement to WHAM because it offers high spatial resolution, compromising spectral resolution. When combined the two surveys will yield a map of absolute Hα surface brightness over the entire northern hemisphere with approximately arc minute resolution.

Some preliminary findings of the Virginia Tech Spectral-Line Survey include the discovery of a supershell detected in Hα in the region of W4[4]. Previously thought to be an open galactic chimney, the W4 supershell is a large bubble of hot ionized hydrogen gas rising from the galactic plane. The bubble has been inflated by energetic outflow of gases from a

cluster of nine hot O-type stars in the galactic plane. The star cluster inside also acts to ionize the thin shell. This bubble, extending 230 pc across, is one example of the complex and dynamic environment of the interstellar medium. Other results are the product of deep Hα observations. These observations rule out the contribution of any significant foreground galactic free-free emission to anisotropies in the cosmic microwave background[19].

The full scope of scientific data that VTSS will provide to the astronomical community can not be foreseen at this time. But the completed survey will provide an invaluable resource to help astronomers gain new insight into the warm interstellar medium and the entire galactic environment.

Chapter 2

Observatory Automation

2.1 An Introduction to Observatory Automation

An automatic telescope, as defined by Boyd, Genet and Hall[2], is a system "...used for the routine gathering of astronomical scientific data that normally operates without human assistance." Many automatic telescopes that fit this description are now routinely used for astronomical measurements and observations around the world. Because of the nature of each particular study automatic observatory systems are highly specialized. Each one is designed to follow continuous and repetitive algorithms to complete observations without human involvement. It is the repetitive character of these studies that makes automation advantageous.

Personal computers with powerful processing capability are readily accessible and versatile enough to allow complete automation of an observatory. Certainly a computer that can perform necessary routine tasks and make basic decisions in the place of a human is both time and cost efficient. An inherent characteristic of a computer controlled system is a well defined and highly repeatable standard operating procedure. This is a desirable trait when performing scientific measurements and observations.

Automated observatory systems have been utilized for several types of studies. Automatic telescopes, such as the Automatic Photoelectric Telescope described by Genet et al.[9] have been used for photometric measurements of variable stars. The Indiana CCD Automated Telescope produces photometry measurements on interacting binary stars to monitor mass transfer rates[12]. The regular and repetitive observations that photometry studies require is ideally suited for an automated system. Other automated studies include monitoring supernovae, active galactic nuclei, comets and asteroids[20, 7]. Much can be learned by carefully and systematically measuring the light curves of these objects as they change over time. With a more general design, the Berkeley Automatic Imaging Telescope(BAIT)[18, 20] allows subscribed users to submit observing requests over the Internet. The system

automatically schedules the submitted requests and completes each observation. Then the astronomer can retrieve the results from the system over the Internet. BAIT is currently used for a variety of different projects from multiple groups and for undergraduate instruction[18].

It is obvious that well made automated systems are beneficial. Each system is tailored to meet specific research criteria. To meet the unique goals of each study automated observatory systems become highly specialized research tools.

2.2 Automating SLIC

The SLIC observatory was designed and built by Dr. Brian Dennison, Dr. John Simonetti, and Dr. Greg Topasna. The process of automating SLIC involves updating the existing SLIC observatory system and eliminating as much humanoid involvement as possible. The SLIC observatory is a complicated system, it consists of many components that have to work together consistently and reliably. Each of these specialized pieces plays an important role in the proper execution of the scientific survey. In the past all the equipment was controlled and operated by the observer. This person had to be present for anything to proceed at all. The observer was there to make decisions, follow procedures, monitor progress, maintain proper operation and keep records of every task performed. The astronomer was necessary in order to continue observing.

The final system is able to do all these things automatically. The new system complies to a well defined set of rules designed to follow the regular sequence of observations and to accommodate unpredictable situations. This set of rules and decisions is in the form of a program running on a PC. This program is called SLICAR (Spectral Line Imaging Camera Automation Routine). By following this programmed set of instructions SLICAR will always know what it should do, and when it should do it. This program must be able to communicate with and control various electronic and mechanical components in order to complete its duties. The program must be able to keep an accurate and explicit record of tasks completed. It must know when things are working properly, and what to do if there is a malfunction. The computer program will completely take the place of the human observer.

One of the many tasks SLICAR will take over is that of imaging. First of all it should know what fields it has to observe. Each field has numerous pieces of information associated with it such as its right ascension and declination coordinates, the filter to be used, how to focus the lens and the length of exposure. Other important information SLICAR must know includes the name of the field, when the field is visible from the observatory location, when it should be observed and when it is actually observed (these could potentially be different). Once SLICAR knows this information it is able to control the telescope mount to point the camera in the right direction and track a field throughout the exposure process. The computer can alter the focus of the lens, and select the correct filter to use. The program operates the camera to expose the CCD and record an image. After exposure the image is saved in a

systematic way with an appropriate file name and directory on the computer. Once that field is finished SLICAR moves on to the next field and repeats the process throughout the night. The various individual systems that work together just to take a picture include the filter changing device, a focusing mechanism, the camera mount and the CCD camera.

Before SLICAR can initiate observations it has to know when it can start observing. The program knows when the Sun is far enough below the horizon to begin the observing session. It also knows when twilight begins in the morning so it can plan the observing session to fit within this window. In addition, all fields are imaged when the Moon is not visible in the sky because the Moon is just as detrimental to these observations as the Sun. These procedures, and others, require the computer to keep accurate time. Access to the Internet will allow it to accurately update its internal clock on a regular basis.

Of course every image will be completely meaningless unless the roof of the building is taken out of the way. When it is time to start SLICAR will open the observatory roof so the camera is exposed to the sky. Then, at the end of the session, SLICAR will close the roof to protect the instruments inside the building. Another occasion to close the roof is when the weather turns bad. Not only can observations not proceed when the sky clouds over, but the instruments must be protected from the elements. Therefore there must be a way to constantly and reliably monitor the current weather conditions. In this way SLICAR will know when it begins to get cloudy and suspend observations by closing the roof.

The roof is a very important part because it protects the camera and other equipment from the outside environment. For this reason it must maintain proper operation in any situation. The roof must be able to roll closed even when power is lost to the observatory. Therefore, the roof and the main computer will have an independent, backup power supply. In the case of a power failure the system will switch to backup power and operate long enough to close the roof and shut down for the night. This is to prevent draining the backup power completely. As a safety measure the roof should also be able to operate independently. If the computer should ever crash or lose all power the roof will automatically close regardless of any other operations in progress. In the event that the roof malfunctions and does not close SLICAR will notify an observatory administrator immediately by phone so that emergency action can be taken to correct any problems.

For the CCD camera to operate correctly its temperature is maintained at -110.5°C. This is done with a liquid nitrogen cooling system on the camera. The temperature of the camera must be kept constant throughout the observing process to insure consistent results. Cooling the camera is done by periodically filling a dewar on the camera with liquid nitrogen, before the liquid nitrogen evaporates completely it is time for a refill. It is imperative for SLICAR to be able to keep the camera at a temperature of -110.5°C by knowing how and when to fill the camera dewar. The initial fill of the camera dewar in the beginning of observations takes a much longer time than the refills because the camera has to cool down from the ambient temperature to -110.5°C. Therefore the initial fill of the dewar begins a specified amount of time before observations will start.

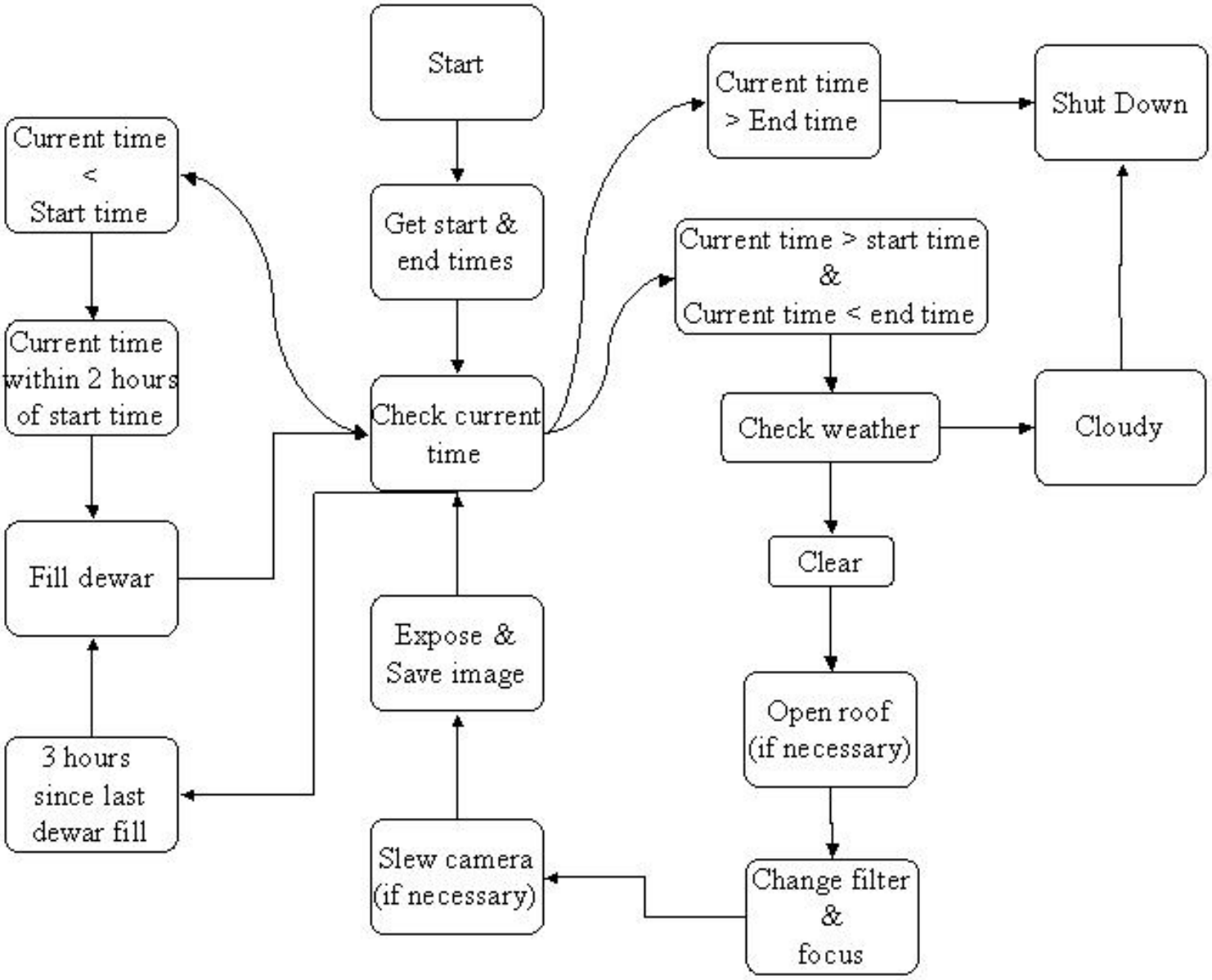
Another crucial aspect to this project is a consistent and current record of all operations that transpire during each observing session. An observation log will be written by SLICAR for every night of observation. It will write to this file all tasks that were attempted, and whether the attempt was successful or not. Along with each log entry will be the date and time of that particular event so that the progress of observations can be monitored. The names and times of all the fields observed, along with important information such as the filter and focus used, exposure time and coordinates, will be entered into the observing log. Entries will include operations such as opening and closing the roof, changing the filter and focus, refilling the camera dewar and slewing the mount. Starting times and ending times will be written to the observation log. Any malfunctions that occur will also be included in the observing record. All these pieces of information are important for standard record keeping, image processing and analysis of the efficiency of the observatory.

A basic observing night proceeds as follows. The computer finds the starting time and ending time for the current session. Then the system waits, constantly monitoring the time. A predetermined amount of time before observing begins the camera cool-down procedure is initiated. The exact time to begin this procedure is resolved so that the time when the camera is completely cooled precedes the start of the observing session. When it is time to start SLICAR finds the list of fields it is supposed to observe that night. The computer checks the weather monitoring device to see if conditions are suitable for observing. If the weather monitor indicates that it is cloudy then the observations do not start. If the sky is clear SLICAR opens the observatory roof. Then it slews the telescope so it is pointing to the correct field and sets the correct focus and filter. The computer records an image of the desired exposure time, saves that image and moves on to the next field. After each image that is completed SLICAR checks the weather monitor to see if observing conditions are still suitable to continue. At a specific time period after the last fill the camera dewar needs to be refilled with liquid nitrogen, and at that time the fill procedure is initiated. When the liquid nitrogen refill is complete SLICAR continues with observations. The computer continues running things in this manner until the end of the observing session which is dictated by the Moon or Sun rising, when it shuts down and waits for the next night.

2.3 SLICAR

The main control program running the observing procedure, called SLICAR (Spectral Line Imaging Camera Automation Routine), was written in collaboration with Phillip Nelson. This program has been coded in PERL. SLICAR reads an Observing File made by the user. The Observing File contains a list of images to observe and can include other inputted commands for SLICAR to follow. A simplified flow diagram of the algorithm which SLICAR implements can be found in figure 2.1. A more detailed illustration of the observing procedure can be found in section 4.2. Appendix A contains the source code for SLICAR.

Figure 2.1: A simplified flow diagram which SLICAR follows.



Chapter 3

Devices for Observatory Automation

3.1 Overview

As stated previously the VTSS has been partially completed during earlier work. The equipment used for this earlier work is still in working order and could be used, as is, to complete the survey. In addition, for each observing session a human must be present throughout the night to conduct the proper operating procedures. Many prime observing hours (dark, clear skies) could be missed due to an observer not being available at all times. If the observatory was automated to conduct the survey with minimal need for human interaction then the amount of time observing during good conditions could be maximized. This is the motivation for automating the SLIC observatory.

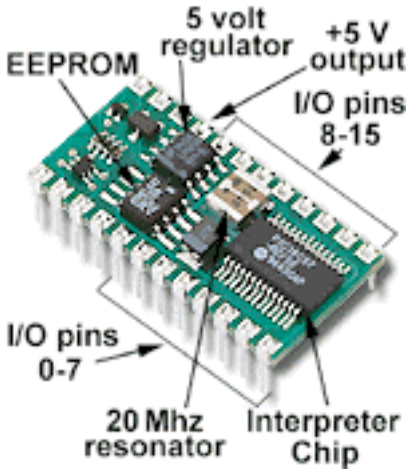
To automate SLIC a number of components were designed and added to the existing system. In addition, this upgrade includes software written to allow a computer to control and communicate with each piece. SLICAR was written in PERL (Practical Extraction and Report Language) which is a versatile programming language that is freely available for various operating systems. This system is running Windows 2000 with a Pentium 4 processor. Each external component is connected to a serial port, through which SLICAR can send and receive communications. The CCD camera is connected through a parallel port. The various hardware devices that make up the system are a roll-off roof, a cryogenically cooled CCD, a liquid nitrogen(LN_2) auto fill device, a lens focusing mechanism, a four-position filter wheel, an equatorial mount and a cloud monitoring system.

3.2 The BASIC Stamp

3.2.1 How it works

An integral part of the automated design is the incorporation of microcontrollers called BASIC Stamps. For this project two BASIC Stamps are utilized to allow communication with the LN_2 refill system (section 3.5), the focuser(section 3.4), the filter wheel (section 3.3) and the roll-off roof (section 3.7). One Stamp is used in conjunction with the roll-off roof for safety reasons detailed in section 3.7. The second Stamp is used to communicate with and control the focus mechanism, the filter wheel and the LN_2 filling device. The model of BASIC Stamp used in the design of SLIC is the BSIIe. The BASIC Stamp is a simple, reprogrammable computer produced by Parallax Incorporated. These devices can directly interface with TTL level devices through programmable input and output (I/O) pins. See figure 3.1 for a picture of BASIC Stamp model BSIIe.

Figure 3.1: The BASIC Stamp microcontroller. The input/output pins are labeled 0-15. Copyright Parallax, Inc.



The BASIC Stamp, or Stamp, runs a simplified and customized version of the BASIC programming language called Parallax BASIC or PBASIC. A program can be compiled and stored in the Stamps internal memory through the serial port, and will remain in memory until it is reprogrammed. The program stored on the Stamp will remain in memory even when power to the Stamp is turned off. When power is given to the Stamp the program inside will start from the beginning automatically, but all information contained in variables is lost. This aspect of resetting the Stamp is important for the roof system (section 3.7) and the focusing system (section 3.4).

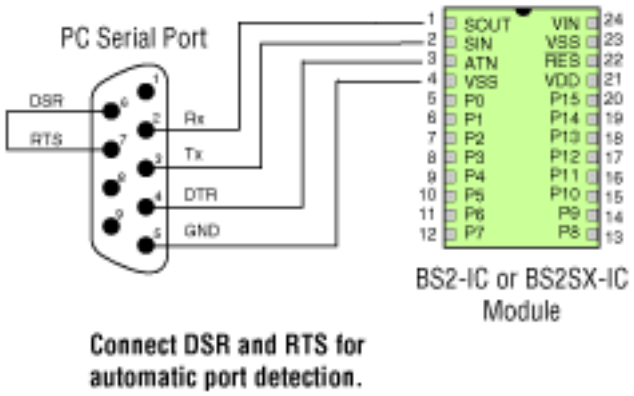
The BASIC Stamp has sixteen general I/O pins. Each pin is individually addressable by its designated name, P0-P15. Each pin is considered a bit of information and can be either 0 (low) or 1 (high). Sets of pins, or bits, are also addressable. The smallest set is of four

adjacent pins, such as pins 0, 1, 2, and 3. A set of four bits is called a nibble. The four possible nibbles are pins 0-3, 4-7, 8-11, and 12-15. They are named A, B, C, and D respectively. A set of two nibbles is called a byte. The set containing nibble A and B together is called L (for low), C and D combined is called H (for high). Putting both bytes together makes a word, and is addressable by the name S.

The I/O pins are used to connect the Stamp to a particular device. As a simple example, it might be desirable to monitor the temperature of an aquarium. In particular, a temperature sensor can be used in conjunction with a Stamp to alert the aquarium operator if the temperature of the aquarium falls below 25°C. A properly calibrated thermistor can be connected to a pin of the Stamp, and the Stamp can watch the state of that pin. When the temperature of the aquarium drops below 25°C the resistance of the thermistor will change enough to alter the state of the pin to which it is connected. The Stamp program monitors this pin, and detects when the state changes. When this change is detected the Stamp can send output to a different pin. Attached to the output pin may be a buzzer or an LED which can alert the operator of the temperature change. A very similar device is used for for the LN_2 dewar refill system and is described in section 3.5. This is an example of how to use individual pins or bits, and example of the use of nibbles can be seen in section 3.3.

The Stamp, connected to the serial port of a PC, can communicate directly with SLICAR. In this way the Stamp allows the main program to detect the status of individual mechanical and electronic components. SLICAR can then instruct the Stamp to perform a certain task with a device. The Stamp can also report the status of a system to SLICAR. Figure 3.2 illustrates the proper connection between the PC serial port and the Stamp.

Figure 3.2: This diagram shows how to connect the serial port to the BASIC Stamp. In this diagram the I/O pins are named P0-P15. Copyright Parallax, Inc.



When the Stamp is connected to a serial port in the way shown in figure 3.2 the PBASIC Stamp program can send and receive information through the serial port using these commands:

For sending information:

```
serout 16,16780,[DEC1 info_var]
```

For receiving information:

```
serin 16,16780,[DEC1 info_var]
```

The commands `serin` and `serout` come with various options and settings. The first number, `16`, refers to the specific pin designated as the serial I/O pin. The second number, `16780`, sets the information transfer rate, or baud rate. The code inside the brackets is the actual information being sent. `DEC1` tells the Stamp it is to send, or receive, 1 character of type decimal, and `info_var` is the name of the variable that contains the information that is to be sent or received.

3.2.2 SLICAR and the BASIC Stamp

SLICAR sends and receives information through the serial port with the following commands:

For sending information:

```
$port_name− >write($command);
($count_in,$echo)=$port_name− >read($n);
```

For receiving information:

```
($count_in,$info)=$port_name− >read($n);
```

In the above PERL commands $port_name is the name of the serial port through which data is sent and received. This name is set while configuring the port in SLICAR. The second part of the send command tells SLICAR to write the variable called $command to the serial port. The second line in the write sequence is reading back an echo of the command just sent. Reading the echo is necessary for the command to be sent and is extremely useful for debugging.

The command to receive data from the serial port is identical to the echo because that is exactly what the echo is doing, receiving information from the serial port. The command reads in $n number of characters from the port called $port_name and stores it in the variable called $info.

It is necessary to initialize the serial ports in SLICAR before communication is possible. The first step is load the module library which allows serial port communication. This is carried out with the command:

```
use Win32::SerialPort qw( :STAT 0.19 );
```

Next the individual serial port can be opened and named with:

```
$port_name=new Win32::SerialPort ("COM1");
```

Finally the settings for the serial are written:

```
$port_name->baudrate(9600) | die "bad baudrate";
$port_name->parity('none') | die "bad parity";
$port_name->databits(8) | die "bad databits";
$port_name->stopbits(1) | die "bad stopbits";
$port_name->read_char_time(0);
$port_name->read_const_time(1000);
$port_name->read_interval(0);
$port_name->write_char_time(0);
$port_name->write_const_time(3000);
$port_name->handshake("none");
$port_name->write_settings | undef $port_name;
```

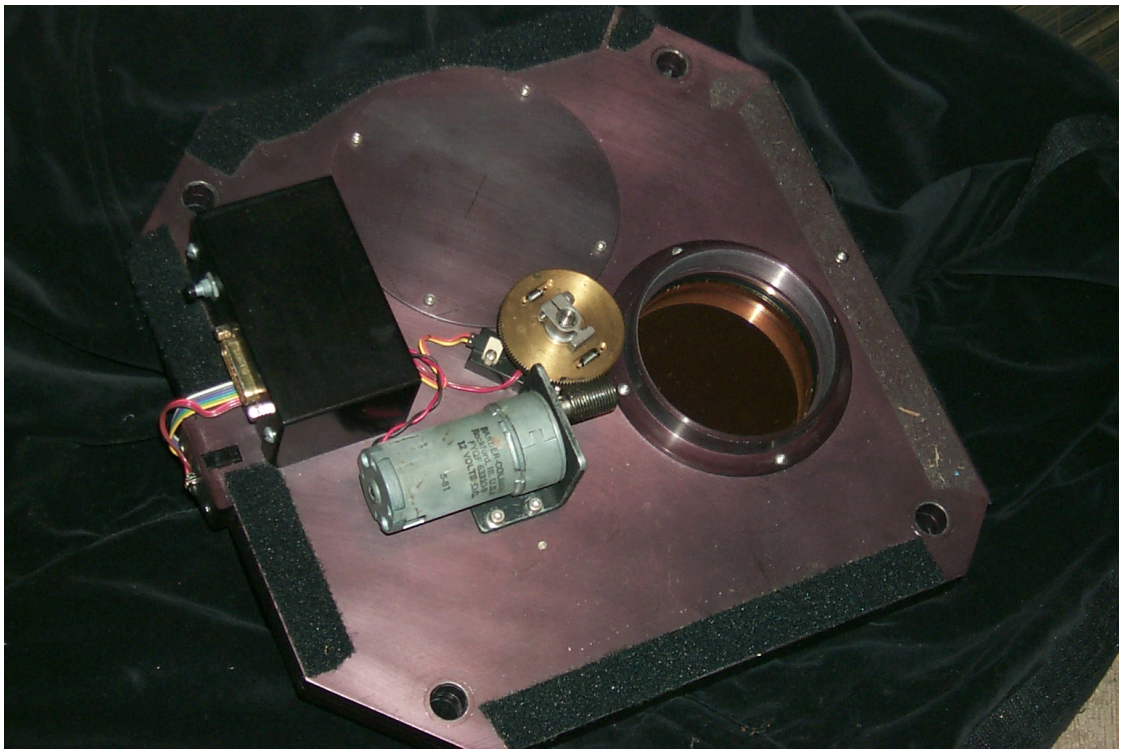
SLICAR can change these settings while running if that is required. Communication through the serial can begin after the settings have been written.

3.3 The Filter Wheel

3.3.1 A Brief Introduction

The filter wheel, designed and built previous to this undertaking, allows easy interchanging of filters for observations at different wavelengths. Each filter is fixed on a rotating wheel inside an aluminum housing to protect them when not in use. A circular window is cut through the housing to allow the passage of light. A shaft through the axis of the wheel protrudes to the outside of the housing, and on it is mounted a small gear. This gear is meshed with a worm gear driven by a small electric motor. When the electric motor is activated it rotates the filter wheel inside. An image of the filter wheel can be seen in figure 3.3. Inside the housing a sensor detects when a filter is in proper alignment with the window. When activated the electric motor runs continuously until the sensor sees the next filter, then it is deactivated. A control box displays the current filter (1-4) that is engaged with a 7

Figure 3.3: The filter wheel. Four different filters can be selected. One filter can be seen here. Also in this photo is the brass gear attached to a rotating shaft and the electric motor with a worm gear meshed to the brass gear.



segment LED display. Prior to automation the observer would push a button on the front of the filter wheel control box to advance the filter wheel. By knowing which filter is in which position one could observe with the desired filter without difficulty.

The filter wheel was made automatic while still keeping this existing system intact. This was done by incorporating a BASIC Stamp into the design. Now the BASIC Stamp performs the tasks of reading the 7 segment display and advancing the filter wheel. The Stamp carries out these operations through its general I/O pins. One output pin acts as the button, and three inputs read from the 7 segment display. By reading the 7 segment display the Stamp knows which filter is place, and the Stamp can advance the filter wheel when the right filter is not yet engaged.

SLICAR communicates with the Stamp through the serial port to tell the Stamp which filter is needed. The Stamp will then advance the filter wheel until the correct filter is in place. Then the Stamp replies to SLICAR by sending the number of the filter that is engaged.

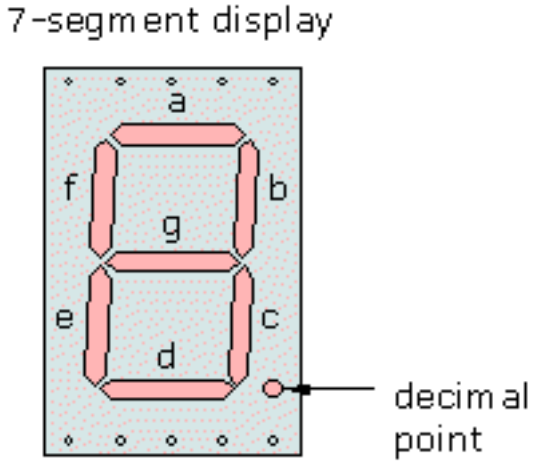
There are several fail-safe algorithms built into SLICAR and the BASIC Stamp program. The first level is in the Stamp program. The Stamp will check and advance the filter wheel five times before it quits and tells SLICAR that it did not reach the filter needed. Secondly,

SLICAR will prompt the Stamp to try again. If this failure sequence occurs three times in a row then the main program detects a malfunction. The third malfunction check is independent of the Stamp. After SLICAR instructs the Stamp to engage a filter, it waits for a response from the Stamp. If no response is received within a specified time SLICAR assumes the Stamp or filter wheel has failed and the observing session is halted. If there is an electronic or mechanical malfunction of the filter wheel SLICAR will realize there is a problem and shut down the system.

3.3.2 Filter Wheel Control

As previously mentioned the BASIC Stamp is able to read the display to determine which filter is currently engaged. It can also advance the filter wheel. The BASIC Stamp reads the display through I/O pins 4, 5, and 6 (named P4, P5, and P6). The pins are connected to individual segments of the seven segment display (see figure 3.4). P4 is connected to

Figure 3.4: The display on the filter wheel control box.



segment g, P5 is connected to segment d, and P6 is connected to segment c. When a particular segment is activated the BASIC Stamp will read the corresponding pin as a low value (0) and when a segment is not lit up the BASIC Stamp will read the corresponding pin as a high value(1). It is more natural to translate a lit up segment as a high value but the electronics between the display and the Stamp invert the output values.

To read the value of the display the Stamp program inputs the value of nibble B (P4, P5, P6, and P7). The value of nibble B is the decimal value which corresponds to the binary input of the four pins. For example, if each pin is 0, then the binary value of B is 0000, and the decimal value of B is 0. If each pin is zero except P4 which is 1 then the binary value of B is 0001 and the corresponding decimal value is 1. Recall that only pins 4, 5, and 6 are connected to the display, but nibble B includes pin 7. When reading from the filter wheel

display pin 7 is always zero. This is because pin 7 is wired to the button which activates the motor to turn the filter wheel. The only time P7 is high is when the Stamp advances the filter wheel to the next filter, then it is only high for a fraction of a second.

The only possible display values are 1, 2, 3, and 4 because it is a four-position filter wheel. To correctly read the display the BASIC Stamp need only know three out of seven segments. The segments c, d , and g give a unique decimal value to the nibble B for each of the numbers displayed, 1 though 4. Table 3.1 below illustrates how each number displayed results in a unique value for the BASIC Stamp input. The number displayed corresponds to the engaged filter. From the table it can be seen that P7 is always zero. The Stamp

Table 3.1: This table illustrates how the BASIC Stamp translates the 7 segment display on the filter wheel into a decimal number

Number	Segment Output			Value of Pin				Inputted	Inputted
Displayed	c	d	g	P7	P6	P5	P4	Binary Value	Decimal Value
1	1	0	0	0	0	1	1	0011	3
2	0	1	1	0	1	0	0	0100	4
3	1	1	1	0	0	0	0	0000	0
4	1	0	1	0	0	1	0	0010	2
None	0	0	0	0	1	1	1	0111	7

determines which filter is in place by checking nibble B. If the correct filter is not engaged the PBASIC program activates the electric motor by toggling the output pin (P7) linked to the button on the control. When the filter wheel is in transit from one filter to the next none of the LED segments are activated. In this case the Stamp will read the nibble B as a 7.

The computer, with SLICAR, tells the BASIC Stamp which filter is needed for a particular observation. The PERL code to send this command is as follows:

```
$ffd_port− >write($filter_goal);
($count_in,$echo)=$ffd_port− >read(1);
```

$ffd_port is the name of the serial port to which the filter wheel Stamp is connected. The information sent to the Stamp is the variable named $filter_goal which has previously been set to the number of the desired filter. The Stamp receives this information via the command:

```
serin 16,16780,[DEC1 filter_goal]
```

The information sent from SLICAR is stored in the value filter_goal. Then the Stamp can

check which filter is engaged by inputting nibble B. The portion of PBASIC code which determines the filter number displayed is as follows:

```
Filter_top
     LOW 7
     in_num = INB
     if in_num = 3 then Filter_1
     if in_num = 4 then Filter_2
     if in_num = 0 then Filter_3
     if in_num = 2 then Filter_4
     if in_num = 7 then Filter_top
```

In this portion of code `Filter_top` is a reference label. A reference label in the PBASIC language is a way to name specific parts of the program. It can be used with an `if` or `goto` statement to have the program jump to the labeled line in the program. `LOW 7` sets the value of pin 7 equal to zero. The third line `in_num = INB` sets the variable called `in_num` equal to the decimal value of nibble B. The last five lines determine which filter is engaged, and go to the appropriate label to continue the algorithm. Using this information the Stamp compares the current filter with `filter_goal` (which it received from SLICAR) and advances the filter wheel until the right filter is in place. The PBASIC code which executes this algorithm is given here in the case when filter number 1 is engaged and `in_num`=3:

```
Filter_1
     filter_num=1
     GOTO check_it

Check_it
     if filter_num=filter_goal then Done
     if filter_num<>filter_goal then Turn_it

Turn_it
     if missed>=5 then filter_failed
     missed=missed+1
     HIGH 7
     Pause 500
     LOW 7
     Goto Filter_Top
```

It can be seen that a counter is employed in this checking routine. This counter counts the number of times the Stamp has advanced the filter wheel after it has been given the command

to choose a particular filter. If the Stamp advances the filter wheel 5 times without reaching the desired filter it proceeds to inform SLICAR that it failed by going to the program label `filter_failed`, which will send a 0 to SLICAR. When SLICAR gets a reply from the Stamp indicating that it did not reach the filter indicated, SLICAR will shut down.

After the Stamp reaches the correct filter it replies to SLICAR by sending the number of the current filter. For added redundancy SLICAR proceeds to double check this value to be sure the correct filter is in place. If SLICAR determines that the correct filter is not in place it will prompt the Stamp to try again by resending the command to select a filter. If this failure occurs three times in a row then a malfunction in the filter wheel is indicated and the observing session is halted. The procedure to receive and check the reply from the Stamp is implemented as follows in SLICAR:

```
($count_in, $filter)=$ffd_port− >read(1);
&tasks_done("filter = $filter");
if ($filter== 0)
      { &tasks_done("**** FILTER WHEEL MALFUNCTION! ****");
      &shut_down;}
elsif ($filter==$filter_goal)
      { $missed=0; }
else { &tasks_done("Did not reach the desired filter.");
      $missed++;
      if ($missed>=3)
            { &tasks_done("*** FILTER WHEEL MALFUNCTION! ***");
            &shut_down;}
      &tasks_done("Trying again to reach filter...");
      goto Filter_top;
      }
```

The &tasks_done("..."); statements are commands that record all important information to a log file.

3.4 The Focusing Mechanism

3.4.1 Automatic Lens Focusing

It is necessary to be able to change the focus of the lens while observing because the filters used are interference filters. These filters slightly focus or defocus incoming light, therefore, each filter requires the lens to be refocused to get the sharpest possible image. Each filter has associated with it an optimum position to focus the lens.

The old lens focusing unit used a circular disc placed on the shaft of an electric motor. Attached around the circumference of the disc was a rubber O-ring. This device was mounted on a spring loaded lever so that the rubber edge of the disc was in constant contact with the knurled grip of the camera lens. When the electric motor rotated the disc, the friction between the rubber edge of the disc and the grip of the lens caused the lens to rotate, thereby focusing the lens. In order to keep track of the angular position of the lens an optical encoder was also used. This encoder had a similar disc on it that was also in contact with the lens. When the lens rotated it pushed the encoder disc measuring the angular rotation of the lens.

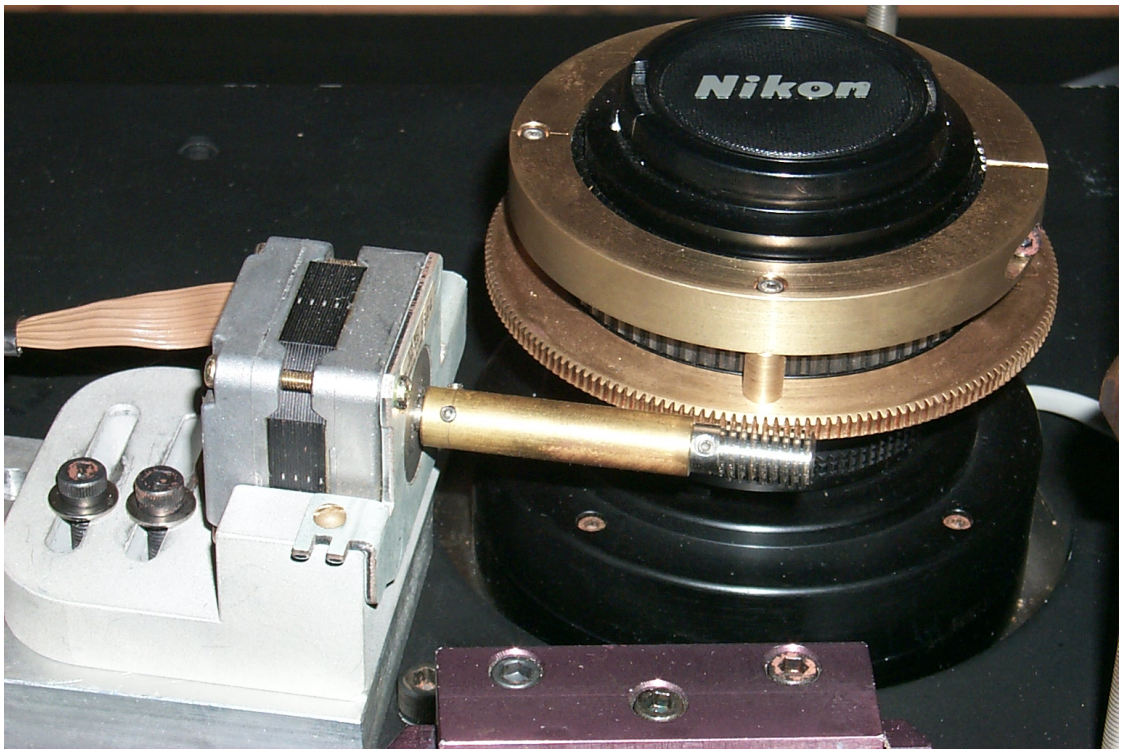
The encoder system of focusing also included a control box located in the control room. It displayed a digital read out from the encoder in arbitrary units. Having already worked out and recorded the best focus position, the observer could push a button to rotate the disc until the optical encoder displayed the correct value and therefore the correct focus.

There were some problems with this method. Rotation of the lens, therefore focusing, depended on constant contact between the lens and both discs. But the rubber edges were subject to slipping. If the disc on the electric motor slipped it would not cause a problem; since it did not affect the encoder measurement in any way. But if the encoder disc were to slip then there would be no way to know exactly how much angular rotation occurred. Also the rubber edges on both discs slowly deteriorated over time, and the system became more prone to slipping. Therefore the focusing device was redesigned to avoid these problems.

The goal for the design of the new focusing mechanism was to produce a system that was more accurate, reliable, and durable than the old system. All three goals were attained with the following setup. A commercially made brass gear and mating steel worm drive are employed for the new system. A hole is cut through the gear and it is centered and attached around the knurled rubber hand grip of the lens (see figure 3.5). Meshed with this gear is a worm gear. The worm gear is driven by a disc drive stepper motor taken from a PC. The stepper motor is mounted so the worm is centered vertically on the gear. Horizontally, the motor and worm combination is adjustable in two dimensions to allow optimum positioning of the worm and gear. Once the worm is meshed correctly with the gear the motor is tightened down with set screws to prevent it from shifting. The stepper motor is rotated by periodically sending pulses to it. Each pulse rotates the stepper motor shaft, and worm gear, $\frac{1}{200}$ of a full rotation. It can rotate clockwise and counterclockwise. The BASIC Stamp used with the filter wheel is also used with focusing mechanism.

The Stamp also keeps track of how many steps the motor has turned with a simple position algorithm. Upon start up of the Stamp it assumes the motor is in position number 0. This is the home position for the motor. When the Stamp is reset the program starts from the beginning (see section 3.2). The BASIC Stamp programming does not allow the motor to rotate to negative positions. Attempting to rotate the motor to negative positions would put mechanical stress on the camera lens because it is trying to focus the lens beyond infinity. SLICAR prompts the Stamp to rotate the motor to a new position. The new position is given as the absolute number of steps from 0. The Stamp calculates the direction and number of

Figure 3.5: The focusing mechanism. A worm gear is attached to the shaft of a stepper motor which is mounted on the front of the camera cradle. The worm drive is carefully meshed with the gear mounted on the lens. The position of the worm drive can be adjusted by loosening two pairs of set screws. Only one pair of set screws can be seen in this photo, to the left of the stepper motor.



steps needed to get from its current position to the new position. Then it pulses the motor that number of steps in the correct direction, and sets the current position equal to the new position. In this way the Stamp keeps track of the stepper motor position at all times.

Of course there are some limitations to this method as well. As explained in section 3.2, when the Stamp loses power the PBASIC program resets and all memory of variables is lost. If the system loses power in the middle of observing the position of the focuser will be lost because this value is stored in the Stamp memory. The worm must be disengaged from the gear, then the lens reset to 0. After turning the Stamp and focuser on, the gear and worm can be mated again. It is important that the focuser be turned on *before* the gears are mated, because when it is turned on, the shaft automatically rotates a small amount. It is for this reason that this focusing system is connected to a backup power supply, called a UPS (Uninterruptable Power Supply), to prevent power loss. This idea of the Stamp resetting after a power loss does not cause a large problem because power loss will require some other systems to be restarted as well.

Another problem occurs when meshing the worm and gear themselves. They must be carefully placed in contact, with the worm tangent to the gear, and the teeth must not be jammed

together. If the positioning of the worm is not correct it could cause the gear to bind. If the gear binds, the motor will fail to turn, and the position will be unreliable. This problem can be prevented by carefully aligning the worm with the gear whenever realignment is necessary.

By realizing these potential problems and preventing them, this new focusing system becomes much more reliable then the old method. The precision machined parts are much more durable and will not degrade significantly over time.

This new method of focusing is much more accurate. The old system, where measurements were made by the optical encoder, rotated the lens $\sim 10^\circ$ while measuring approximately 300 units or steps (the units used to measure the rotation were arbitrary, given by the digital display box). This is 30 steps per degree of angular rotation of the lens, or 0.03°/step. The new system is using a 64 pitch gear with a pitch diameter = 5.625 inches. This translates into a gear with 360 total teeth, or 1 tooth per degree of rotation. One complete rotation of the worm will advance one tooth on the gear. Therefore one complete revolution of the worm is equivalent to one degree of angular rotation of the lens. Since there are 200 steps per one revolution of the worm, the new system is accurate to 0.005°/step. This is an order of magnitude more precise than the old system.

There is backlash associated with this type of gearing system. Backlash is the inherent looseness, or play, in a mechanical system that arises because of the need for space between the teeth of two meshing gears. With this setup the backlash is $\sim$0.001 inch (this value provided by the manufacturer), which translates to around 5 steps. So the maximum backlash is $\frac{1}{40}^\circ$, which is still smaller than one step of the old system. Backlash can be completely avoided by always arriving at the desired position from the same direction. The backlash associated with old system is difficult to quantify and consequently was not measured or documented.

There are algorithms built into SLICAR to account for focusing malfunctions. After SLICAR prompts the Stamp to rotate the lens, it waits for a reply. If the Stamp does not respond within a predetermined time period then SLICAR knows it has malfunctioned and the shutdown procedure commences.

3.4.2 Controlling the Focusing Mechanism

The focusing system incorporates the same BASIC Stamp that is used for the filter wheel. The stepper motor is connected through a control board to the Stamp. The control board regulates communication between the BASIC Stamp and the stepper motor. The Stamp uses four output pins to control the motor. The bits used are P1, P2, P3, and P15. Pin 15 is always set equal to zero by the Stamp. Output to P1 selects the stepper motor. (It is possible to have 2 motors connected to the control board. P1 would select one motor, another pin could be used to select the other motor. There is only a single motor in this case, so it is always selected.) P2 allows the Stamp to select the direction the motor will rotate. Finally, pin 3 is used to send pulses to the stepper motor causing the worm shaft to

rotate.

SLICAR prompts the Stamp to turn the motor by sending a four digit number to the Stamp. This four digit command is actually a position to which the motor should turn. The position is given as the absolute number of steps from zero, or home. The PERL command which sends this number to the Stamp looks like this:

```
$ffd_port− >write($new_pos);
($count_in, $echo)=$ffd_port− >read(4);
```

Note the value given to the $read command (which receives the echo of the command just sent) is 4 indicating that the data being read is 4 characters long.

Given this number the Stamp calculates the direction and number of steps it needs to turn to reach the new position. The Stamp receives the new position with the standard statement:

```
serin 16,16780,[DEC4 newpos]
```

Note the qualifier `DEC4`, which indicates that the incoming data is a decimal value containing four characters. The new position is stored in the variable called `newpos`. The Stamp program proceeds to calculate the direction and number of steps:

```
DoMove:
if newpos=currentpos then DoSteps_exit
if newpos>currentpos then StepUp

     high driveDIR
     steps=currentpos−newpos
     goto DoSteps_move

StepUp:
     low driveDIR
     steps=newpos−currentpos

DoSteps_Move
     for i=1 to steps
          pulsout driveSTEP,2000
     next

currentpos=newpos
```

```
DoSteps_exit
     return
```

Where `currentpos` is the value of the current position of the motor. The pins 2 and 3 are named `driveDIR` and `driveSTEP` respectively. The command `pulsout driveSTEP, 2000` sends one pulse to the stepper (`2000` refers to the duration of the pulse and therefore the speed that the stepper rotates). The direction of rotation is determined by the `if` statement on the second line. Next, the number of steps to get to the new position is calculated and that value is stored in `steps`. Then, at the label `DoSteps_Move` the program repeats a loop `steps` times. Each time through the loop it pulses the stepper once. When this loop is finished the current position is reset to reflect the current position of the stepper and stored in `currentpos`. Finally, the program exits this routine.

Immediately after the program exits the stepping routine it sends the value of the current position to SLICAR:

```
serout 16,16780,[DEC4 newpos]
```

This is to inform SLICAR that the Stamp has finished focusing. SLICAR receives and processes this data with these commands:

```
($count_in, $pos_in)= $ffd_port− >read(4);
if (length($pos_in)<4 )
     { &tasks_done("*** FOCUSER MALFUNCTION! ****");
     $focus_error="error";
     &shut_down;
     }
```

The $ffd_port− >read(4) command will wait approximately 10 minutes for a reply from the Stamp. This wait time is set when configuring the serial port (see section 3.2). The if statement checks the length of the data received. If the length is less than 4 characters long then the Stamp did not send the correct information. This procedure is to check for malfunctions in the focusing system. If the focusing mechanism breaks it will not be able to send the data back to SLICAR. SLICAR detects that it has not received anything and will indicate a malfunction. In the case of a malfunction, observing cannot continue so SLICAR will shut the system down.

3.4.3 Determining Accurate Focus

Since the value for each focus need only be found once, determining the best focus for each filter is not an automated procedure. At most the focus might need to be checked or reset

once a month.

To find the best focus for one particular filter, first the filter is put in place using the filter wheel. Next the camera is focused to some arbitrary value, an image is acquired through the PMIS software (see section 3.9). The image is viewed and a bright, unsaturated star in the field is chosen. The pixel coordinates for the center of the star are recorded. The image is converted into an ASCII file which contains the value of each pixel in a text format. Now a program, written by Dr. John Simonetti, is run by the user which will calculate the half-flux diameter of the chosen star, after entering the name of the ASCII file and the pixel coordinates for the center of the star. The half-flux diameter of the star is recorded. The focus is changed slightly, and the process is repeated using the same star.

The half-flux diameter is a measure of the apparent size and shape of the star in the image. The best focus is attained when this value is minimized, thereby reaching the sharpest focus of the image. The half-flux diameter is calculated by first identifying the center of the star with a centroid calculation. Once the center is found a series of concentric annuli are are centered on the star. The total flux inside an annulus is found by summing the pixel values within the ring. The total flux for the star is found by summing the flux for each annulus. The half-flux diameter is found by determining the size of the ring within which is contained exactly half of the total star flux.

For a star that is out of focus the half-flux diameter will be large. When the camera is focussed the half-flux diameter will be minimized. By taking a series of images at different values of focus and plotting the half flux diameter of a star, the optimum focus can be found.

As it turns out, finding the focus is not as simple as that. The matter is made more complicated by the fact that the value of best focus is different in different parts of the image. Mainly, when a star at the center of the image is in focus, a star at the edge of the field will be slightly out of focus. The same is true for when a star is in focus at the edge of the field, the center of the image will be out of focus. This effect is due to having a wide field of view, compounded by using interference filters. For this reason the best focus for each filter is a compromise between having the image focused at center, and focused at the edge.

3.5 The CCD Dewar Refill System

3.5.1 The Design

In order to observe with the best sensitivity the optimum temperature for the CCD camera to operate at is -110.5°C. The camera head is a Photometrics CH260. Cooling the system is achieved with a liquid nitrogen (LN_2) system. A small LN_2 dewar is attached to the camera and is filled with LN_2 from a larger resupply dewar (see figure 3.6). The temperature is

monitored and maintained by the Photometrics CE200A Camera Electronics Unit.

Figure 3.6: The LN_2 resupply dewar. A flexible hose from the back of the camera dewar(camera dewar not pictured) is attached to the mechanical valve (blue) which in turn is connected to the resupply dewar.

Periodically the camera dewar needs to be refilled because the LN_2 eventually boils off. The process of filling the dewar starts by pointing the camera in the correct position for filling (pointing the camera is covered in section 3.6). The refill position, pointing directly west towards the horizon, has been chosen to allow a maximum amount of LN_2 into the camera dewar while also being a safe position in terms of keeping cords and hoses untangled. The next step is to open a valve connecting the camera dewar to the resupply tank, through a flexible hose. Pressure inside the supply tank forces liquid nitrogen to flow through the hose and into the camera dewar. When the camera dewar is full, LN_2 leaks out through an overflow outlet on the back of the camera dewar.

To automate this particular component an electronically controlled valve and a thermistor are incorporated into the system. The LN_2 auto fill system also utilizes the same BASIC Stamp microcontroller that is used for the filter wheel and the focusing mechanism. The electric valve is placed in the connection between the LN_2 resupply dewar and the camera dewar (see figure 3.7). The thermistor is placed in the overflow reservoir, which has been added to the overflow outlet, so it can detect the outflow of liquid nitrogen (the thermistor will also be called the LN_2 sensor).

Figure 3.7: Close-up of the electric valve.

The electric valve can be operated by supplying power to it, or cutting its power supply. When the valve is given power it automatically opens, which allows LN_2 to flow from the resupply tank to the camera dewar. When the camera dewar is full the power to the valve is turned off and the valve closes, halting the flow of LN_2.

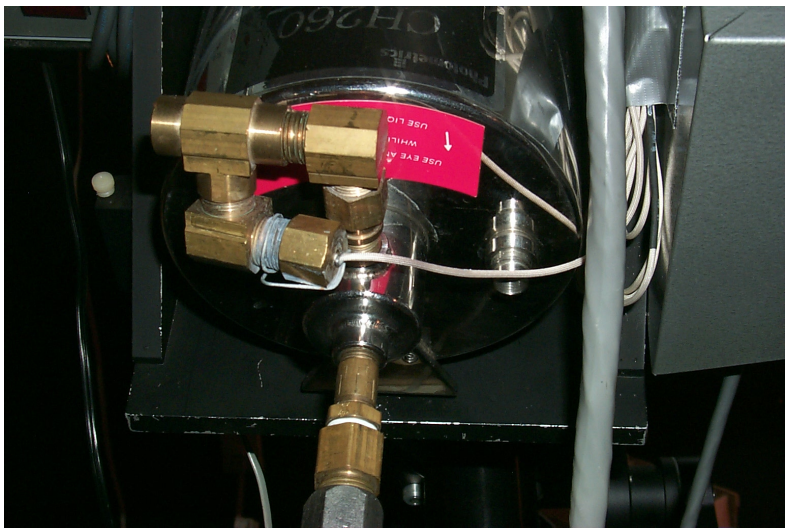
Brass tubing, added to back of the overflow outlet of the camera dewar, acts as an overflow reservoir for liquid nitrogen (see figure 3.8). The shape and orientation of the tubing is made so that, when the camera is in the filling position, the brass pipe acts to trap overflowing liquid nitrogen in a reservoir while allowing nitrogen gas to escape freely. Any moisture that might collect inside the trap while observing is allowed to escape through a small seep hole in the LN_2 collection reservoir.

The LN_2 sensor is placed inside the reservoir. When the dewar is filled LN_2 will spill through the overflow and collect in the trap. The LN_2 level will rise inside the brass pipe until it contacts the LN_2 sensor. By monitoring the thermistor, the LN_2 auto fill system can reliably determine when the refill is complete

The system must still determine when the dewar needs to be refilled. Past experience has shown that a full camera dewar (1.1L of LN_2) will last somewhere between 3-4 hours. Included into SLICAR is a feature which monitors the time and initiates the refill process every three hours (a dewar fill will not interrupt other processes, it will wait until they are finished). Also in the Observing File (which is detailed in section 4.1) a fill command can be placed to force a refill at a certain time during the observing procedure.

The LN_2 auto fill system is not completely automated in that it still requires regular attention. The LN_2 resupply dewar needs to be refilled periodically. This dewar holds 25 liters of liquid nitrogen and this supply lasts for about 2-3 days. Every few days the used resupply

Figure 3.8: Overflow tube and LN_2 sensor. Nitrogen gas escapes from the upper left part of this brass tubing. When the camera dewar is full, liquid nitrogen fills the brass piping and spills into a reservoir. The sensor is inserted into the reservoir through a hole in a brass cap. The white sensor cable can be seen exiting the brass cap in the center of the picture. Also pictured here is the LN_2 refill hose attached to the back of the camera dewar, seen at bottom center.



dewar must be replaced with a second, 25L, resupply dewar which is full. A larger resupply dewar was considered (∼50L) to increase the amount of time between dewar replacements. The mass and bulk of a larger size tank would introduce more problems of its own, so the 25L resupply dewar is preferred. This is not a problem because the entire observatory will need regular checkups to ensure proper operation and peace of mind.

The software also accounts for malfunctions in the LN_2 refill system. There is a time-out routine, similar to the time-out routines for the filter wheel and the focuser, which waits for a statement from the BASIC Stamp indicating that the camera dewar is full. If too much time elapses before this response is received the system is shut down. The LN_2 auto fill system was designed and built by Phillip Nelson

3.5.2 Sensing the Sensor and Valving the Valve

SLICAR dictates when the camera dewar needs to be refilled. The procedure which SLICAR and the BASIC Stamp follow to fill the camera dewar has been coded by Phillip Nelson. When it reaches this time SLICAR will launch the dewar refilling procedure. The dewar refill procedure begins by positioning the camera in the refill position (details of positioning the camera are given in section 3.6). Next, SLICAR will instruct the BASIC Stamp to refill the camera dewar.

The first step in the Stamp program when filling the dewar is to open the electric valve. The

electric valve is connected to a relay which can be switched to open and close the valve. This relay on the valve, connected to P8 of the Stamp, is operated by the Stamp. The Stamp program closes the relay by setting P8 high, which sends power to the valve, thereby opening the valve.

After the valve is open it is the job of the Stamp to monitor the filling process. This system is similar to the example of monitoring the temperature of an aquarium illustrated in section 3.2. A control board is used in conjunction with the LN_2 sensor. The control board, when connected to the Stamp, can tell the Stamp when the temperature of the themistor has reached the temperature of LN_2. Pin 10 of the Stamp is wired to the output of the control board. When filling the camera dewar, the PBASIC program constantly monitors P10. When the control board detects that the thermistor has contacted LN_2 it switches the state of P10. The Stamp then sees P10 change and knows the dewar is filled with LN_2.

The PBASIC code which controls refilling the camera dewar is shown here:

```
HIGH 8
Open:
     PAUSE 500
     pin10=IN10
     SEROUT 16, 16780, [DEC1 pin10]
     SERIN 16, 16780, 100, Continue, [DEC1 fill_status]

Continue:
     IF fill_status=2 THEN Fill_Done
     IF pin10 = 0 THEN Open

Fill_Done:
     LOW 8
```

Note the extra qualifiers contained in this `SERIN` statement. The third number `100` tells this input statement to wait 100 milliseconds for data through the serial port. If no data is received the argument `Continue` instructs the PBASIC program to skip to the position in the program labeled `Continue:`. The `SERIN` statement here allows PERL to interrupt the fill process if necessary. SLICAR monitors the time duration of the filling process. If it detects that the filling process is lasting too long it will interrupt the Stamp, which halts the fill procedure.

The command `HIGH 8` sets the state of P8 to 1, or high, which closes the relay and opens the electric valve. The statement `pin10=IN10` sets the value of the variable `pin10` equal to the state of P10, which will be either high or low. When P10 is low, or zero, then filling of the dewar should continue because the sensor has not yet detected LN_2. So the program returns to label `Open:` to repeat the process.

When P10 switches high the filling procedure is complete because LN_2 has spilled into the collection reservoir and immersed the sensor. At this time the program moves to `Fill_Done:`. Now P8 is set low, opening the relay and closing the valve.

During the process of filling the Stamp program cycles through a loop. Each time through the loop it sends the status of P10 to SLICAR with `SEROUT 16, 16780,` [`DEC1 pin10`]. This statement is to insure SLICAR that the dewar is filling, and to inform the computer when the fill procedure is finished. When P10 switches high, SLICAR will get this information along with the Stamp. When the fill procedure is finished SLICAR can continue with the observing session.

3.6 The Mount

3.6.1 Gemini System

The camera assembly is equatorially mounted on a concrete pier. The mount is a Losmandy G-11 equatorial mount. Control of the mount is through a DC servo motor based computerized positioning system called the Losmandy Gemini System produced by Losmandy Astronomical Products (see figure 3.9). This replaces the older system which is a less accurate stepper-motor-based positioning system on the same G-11 equatorial mount. This mount system is running Gemini level 2 software, version 1.23. The Losmandy Gemini System allows communication with a PC through a built in RS-232 serial connection. Through the serial connection the PC can send commands to the Gemini mount and read information back from the mount. The commands recognized by this system are a subset of the Meade LX-200 serial command set. The Gemini system also accepts a set of native commands called the Gemini Native Command set.

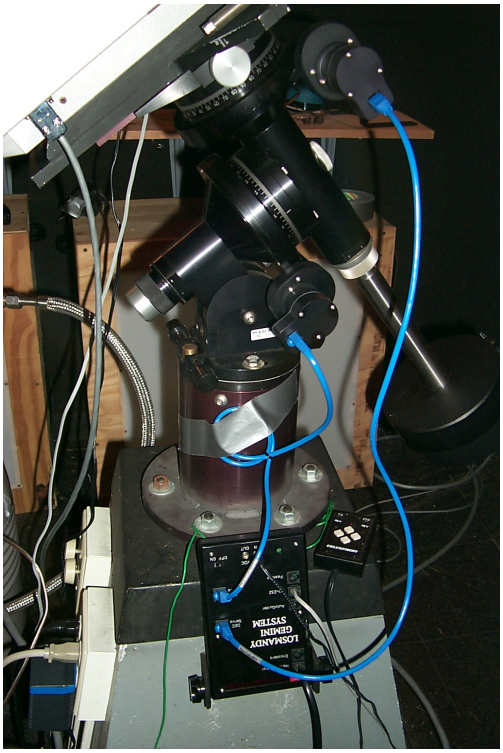
One feature of the Gemini System is very high precision pointing to better than one arcminute. This is achieved by using a pointing model that can use up to 256 alignment stars on either side of the meridian. The pointing model also utilizes built in permanent periodic error correction to eliminate periodic error caused by the slight helical path variation in the shape of the right ascension worm gear. Using the pointing model the Gemini system can accurately track at sidereal rate.

The documentation and manuals for the Gemini system can be found at the Losmandy Astronomical Products website: *http://www.losmandy.com* [22].

3.6.2 Communicating with the Mount

The mount can be operated by the observer by using a hand controller. This hand controller can be used to slew the mount, change settings, and go to selected objects. The mount is

Figure 3.9: The equatorial mount and Gemini system.



indeed designed for use with the presence of an operator. But, for this project the mount is controlled with a PC.

There are various commands available that can be used by SLICAR to operate the Gemini system[21, 22]. These commands must be sent through a serial port to the RS-232 connection on the mount control box. The Gemini commands are sent by SLICAR to the mount with the following statements:

```
$response_count= N;
$mount_string='descriptive text';
$mount_command= 'instruction';
&send_to_mount;
```

The first three statements above are specific settings that depend on the command being sent. The line $mount_command='instruction'; is the actual command that is to be sent to the mount. The word instruction is replaced with the correctly formed command that the Gemini mount will recognize.

Many commands sent to the Gemini will prompt an automatic reply. For example, the user may request the current right ascension to which the mount is pointing. The response

will be a specific number of characters which gives the right ascension. For the purpose of receiving these replies the variable $response_count must be set equal to the exact number of characters expected from the mount. Recall the PERL command which reads data from the serial port:

```
($count_in,$info)=$port_name− >read($n);
```

This command requires the argument $n to be specified. The argument $n sets the number of characters expected through the port. So the variable $response_count, in the above block of PERL code, is set to the number of characters expected as a reply from the mount. In the case when there is not a reply from the mount $response_count can be set to zero. The string $mount_string is for debugging purposes and can be set to anything the user wants. Finally, the statement &send_to_mount; calls the subroutine which sends the command to the mount and receives any reply. The &send_to_mount; subroutine is as follows:

```
sub send_to_mount
    {
    $mount_port− >write($mount_command);
    ($count_in, $response)=$mount_port− >read($response_count);
    # print "$mount_string: $response";
    }
```

It is important to note that in this case there is no echo of the command just sent. Therefore the statement after sending a command to the mount will read the reply from the mount. The echo is a feature that is inherent to the BASIC Stamp. The print statement is commented out because it is only used for troubleshooting.

What follows illustrates the command to the mount which requests the current right ascension. The command to request the right ascension is :GR# (GR stands for Get Right ascension). All standard commands begin with : and end with # (the Gemini Native command set are exceptions to this rule, and are described below). The reply to this command is in the format hh:mm:ss:#, which is 9 characters long. To request the right ascension from the mount:

```
$response_count = 9;
$mount_string='Current_RA';
$mount_command=":GR#";
&send_to_mount;
```

The LX200 commands described above are relatively straightforward. Equipped with these commands, a wide range of operations can be performed. But some actions can not be completed using only the LX200 command set.

One such task that can not be done with these commands is to stop tracking. It is necessary to stop tracking when refilling the dewar, taking flat images (explained in section 4.2), and shutting down. There is another set of commands, called the Gemini Native Command (GNC) set[21, 22], which can be used to perform other operations not included in the LX200 command set. The Gemini Native Command set is less intuitive and more tedious than the LX200 commands.

Each GNC consists of three components. The first is a "set" or "get" character which is a ">"or a "<"symbol respectively. The set character sets a parameter in the mount, whereas the get character retrieves a value from the mount. The second part of each GNC is an identification value. The ID value is a number which corresponds to the type of command being sent. The last piece of the command is called a checksum. The checksum is the value that complicates the command.

The checksum value must be calculated from the transmitted characters. For example, the ID number to set the current tracking rate to "none" (for parking the mount) is 135. The set character is ">". So the transmitted characters are ">135:", in that order. The checksum value is determined from these characters and is also transmitted. The general form of each GNC is < / >ID:checksum#.

The first step in calculating the checksum parameter is to convert the transmitted characters into binary form. Then perform a byte wise XOR operation on the transmitted characters. The XOR is a boolean operation (see table 3.2 for the XOR truth table).

The next step is to clear the highest significant bit of the result (called a modulo 128 operation). Now the binary value of 64 is added. Then, the value of the resulting binary number is translated into the ASCII value. This calculation for the command ">135:", which stops tracking (sets tracking speed to terrestrial), is shown in table 3.3:

Table 3.2: The XOR truth table.

A	B	A XOR B
1	1	0
1	0	1
0	1	1
0	0	0

From table 3.3 the checksum value for the ">135:" command is "s". This means that the full command to set the tracking speed to zero (terrestrial) is: ">135:s". Having worked out

Table 3.3: The Gemini Native Command set translation calculation.

Bytewise XOR operation...

		>	0011 1110			α	0000 1111			β	0011 1100
		1	0011 0001			3	0011 0011			5	0011 0101
$\alpha =$	> XOR 1		0000 1111	$\beta =$	α XOR 3		0011 1100	$\gamma =$	β XOR 5		0000 1001

Add 64...

	γ	0000 1001		δ	0011 0011
	(colon) :	0011 1010		64	0100 0000
$\delta =$	γ XOR :	0011 0011	$\epsilon =$	δ+ 64	0111 0011

$\Uparrow$

Clear this bit. Set the most significant bit equal to zero.

Finally, $\epsilon = 0111\ 0011$, corresponds to the decimal value 115. ASCII value 115 translates into the character (lowercase) s.

the checksum value the full command can be entered directly in SLICAR. This command is sent to the mount in the same manner as the LX200 commands:

```
$response_count=0;
$mount_string='parking sequence initiated';
$mount_command= '>135:s#';
&send_to_mount;
```

It can be seen from the above portion of code that sending this causes no response to be sent by the Gemini.

A total of three GNCs are utilized in controlling the mount. The first, seen above, halts tracking of the mount. The second will set the tracking speed to sidereal. The third GNC requests the current tracking speed from the mount. The three commands and their descriptions are shown in table 3.4.

All Gemini commands, rules, and descriptions have been derived from the Gemini Serial Interface Command Description.

3.6.3 Pointing the Camera

By utilizing the LX200 and the GNC command sets SLICAR can reliably control the mount. The majority of commands going to the mount will be to point the camera to a specific right

Table 3.4: Three important Gemini Native Commands and descriptions.

Command	Description	Gemini Response
<130:t#	Request current tracking rate	135w# (terrestrial) 131s# (sidereal)
>131:w#	Set tracking rate to sidereal	none
>135:s#	Set tracking rate to terrestrial	none

ascension (RA) and declination (DEC) coordinate on the sky. All RA and DEC coordinates used are at epoch 2000. To slew the camera this statement is used:

```
&slew(@coordinates);
```

Where @coordinates is an array containing the RA and DEC coordinates of the desired field. The array is passed to the subroutine &slew. This subroutine points the camera to a field by following this procedure:

```
&SEND_RA;
&SEND_DEC;
&send_name;

$response_count=1;
$mount_string='Sending slew command';
$mount_command= ' :MS#';
&send_to_mount;
```

The first three lines are calls to subroutines. The routine &SEND_RA; sends the RA of the field to the Gemini mount. The &SEND_DEC; subroutine gives the DEC of the field to the mount. Sending the declination also tells the Gemini to select the current field. The coordinates for the selected field are used for subsequent slew commands. The Gemini system will not be able to slew if a field is not selected. The third subroutine, named &SEND_DEC; gives a name to the selected field. The next four statements in the above code form the standard command sending procedure. The command sent in this case is :MS#, which tells the mount to slew to the selected field. The Gemini responds with a 1 if the command was accepted, 0 if not accepted. The &SEND_RA; subroutine called in the above code looks like this:

```
sub SEND_RA
```

```
{ # RA & DEC given as: $RAh:$RAm:$RAs, $DECd:$DECm:$DECs
      $response_count=1;
      $mount_string='RA sent';
      $mount_command=':Sr$RAh:$RAm:$RAs#";
      &send_to_mount;
      if ($response !=1)
      { ++$error;
            print "Error sending RA command.";
            if ($error<= 3)
            { print "Trying again...";
                  ($count_in, $dummy)=$mount_port- >read(10);
                  &SEND_RA;
            }
            else
            { $mount_error='error';
                  &tasks_done("*** MOUNT MALFUNCTION ***");
                  &tasks_done("Error sending RA command to Gemini mount");
                  &shut_down;
            }
      }
      else { $error=0;}
}
```

The first four lines in the routine are again the standard command sending statements. The command that gives the RA to the mount is this :Sr$RAh:$RAm:$RAs#. Where the right ascension hours, minutes, and seconds of the field are $RAh, $RAm, and $RAs respectively.

The remainder of the routine checks the response from the Gemini. If the Gemini received the coordinates, and they are valid, it will respond with a 1. Otherwise the response will be 0. In the case when the response is not equal to 1, the program will try to send the RA command again. If it fails 3 times SLICAR will shut the system down because there is a malfunction. The &SEND_DEC; and &send_name; subroutines are similar to the one illustrated above.

After the :MS# command is sent, the camera will begin slewing to the selected field. Now SLICAR enters the subroutine called &check_slew_reply. The purpose of this subroutine is to interpret the response from the Gemini after this command. There are five possible responses to the :MS# command. The possible responses are exactly as follows:

1) 0
2) 1Object below horizon.#
3) 2Telescope is not aligned.#

4) 2No object selected.#
5) 3Manual control.#

Each reply from the Gemini has a different number of characters, so a series of if statements in &check_slew_reply is designed to filter through the possible replies to determine the exact message.

The reply numbered 1, which is 0, indicates that the slew command was received without error, and the mount is now slewing to the selected field. If reply number 2 is sent to SLICAR, this indicates that the mount has calculated that the selected object is not visible at this time. If this occurs SLICAR moves on to the next field. In the case of response number 3, observing can not continue until the camera has been properly aligned, so SLICAR shuts down. When the mount indicates that no object has been selected this means that the coordinates of the field have not been sent properly. So the slewing procedure is attempted again from the beginning. If this fails 3 times then observing is halted. For the fifth reply observing will also be halted.

When the slew command is received and obeyed by the mount SLICAR continues on to monitor the slew. One reason to monitor the slew is so SLICAR knows when the slew is complete so it can continue with observing. It also monitors the slew to determine if there are any malfunctions.

The subroutine where SLICAR watches the mount as it slews is called &slew_monitor. &slew_monitor loops through a series of commands. The first two commands sent to the mount in this loop request the RA and DEC respectively. The third requests the speed at which the mount is slewing. This procedure determines when the mount is finished slewing by using four temporary variables called $current_RA, $last_RA, $current_DEC, and $last_DEC. The "last" variables store the previous value of the RA and DEC coordinates, and the "current" variables store the most recent value of the RA and DEC coordinates. Each time through the loop the "last" and "current" coordinates are compared with the statement:

```
while (($current_RA ne $last_RA) || ($current_DEC ne $last_DEC))
```

The loop will continue while "last" and "current" are not equal. This loop is executed approximately once per second. Each time through the loop the new coordinates are requested from the mount and replaced, and the last set of the coordinates is shifted into the "last" coordinates, and they are compared. When two consecutive values of the coordinates are identical a counter is incremented. Two consecutive values of the coordinates must be identical for two consecutive iterations of this loop for SLICAR to determine that the mount has stopped slewing. In effect, SLICAR knows the mount is finished slewing when three consecutive values of the RA and DEC coordinates are equal.

The command which requests the slewing speed is used to monitor the slew in more detail. When the mount slews to a destination it begins by ramping up to the *slewing speed*, which is a high speed. When it is pointing close to the target the speed is reduced to *centering speed* so it can hone in on the target more accurately and safely. When the mount reduces to *centering speed*, a timer is activated. If the mount is centering for too long the there may be a malfunction, in which case the slew is halted, and attempted again.

The algorithm to monitor the slewing speed was added in response to a rare malfunction in the operation of the mount. On rare occasions during slewing, the mount reduces to *centering* speed and reverses the slew direction, causing the camera to move away from the selected target at a very slow speed. When speed was added in response to a rare malfunction in the operation of the mount. On rare occasions during slewing, the mount reduces to *centering* speed is detected by SLICAR it starts a timer. If the mount is centering for too long then the SLICAR halts the slew and tries again. This malfunction could cause many problems if not detected. The camera could potentially crash into the pier. Also, a large chunk of the observing session would be wasted as the mount would be slewing for a very long time, while never reaching its target!

Once SLICAR has determined that the mount has finished slewing one more final check occurs before observing continues. This check occurs in a subroutine called &check_RA_DEC. Using this statement:

```
if ( (($current_RAh==$RAh)&&($current_DECd==$DECd))
   && (($current_RAm==$RAm)&&($current_DECm==$DECm)) )
```

&check_RA_DEC simply compares the actual RA and DEC coordinates with the target coordinates that were sent to the mount. If the coordinates are the same then observations can continue. Note that the each coordinate only need to be accurate down to the arcminute. Only rarely do the seconds agree exactly. The error in the angle is typically within 5 arcseconds. This error in pointing is well within the accuracy of the survey as detailed in section 1.3. If the need arises, it is a simple fix to reduce the tolerance in pointing accuracy. But again, the limit is approximately 5 arcseconds.

After the slew has been determined to be successful by this succession of procedures the observations can continue.

3.6.4 Pointing using Altitude and Azimuth Coordinates

Sometimes it is necessary to point to a specific altitude and azimuth instead of right ascension and declination. The dewar refill operation is an example when altitude and azimuth (alt/az) pointing is used (see section 3.5). To put the camera in the refilling position the mount points the camera directly west at the horizon. The alt/az coordinates for this position are always

0° altitude and 270° azimuth. But the mount will only accept RA and DEC. A subroutine, written by Phillip Nelson, which converts alt/az into RA and DEC is included in SLICAR. This subroutine accepts an altitude and an azimuth and returns the current RA and DEC for that position. The program statement looks like this:

```
@coordinates=hor2eq($Alt, $Az);
```

In this line of code @coordinates is the array where the converted RA and DEC will be stored. $Alt and $Az are variables containing the original altitude and azimuth coordinates. The name of the subroutine is hor2eq and looks like this:

```
sub hor2eq
{
my($altitude, $azimuth)=@_;

$altitude=$altitude * $RAD;        $azimuth=$azimuth * $RAD;
$DEC=sin($altitude)*sin($LAT)+cos($altitude)*cos($LAT)*cos($azimuth);
$DEC=atan2($DEC, sqrt(1.0-$DEC**2));
@DEC=decimal_converter($DEC/$RAD);

$x=sin($altitude)-sin($LAT)*sin($DEC);
$y = -sin($azimuth)*cos($LAT)*cos($altitude);
$H_A=atan2($y, $x);
$H_A=$H_A/(15.0*$RAD);

@time=gmtime;
$UT=DECIMAL(1, $time[2], $time[1], $time[0]);
$day=$time[3];
$month=$time[4]+1;
$year4=$time[5]+1900;
$LST=UT2LST($UT);
$RA=$LST-$H_A;
$RA=time_range($RA);
@RA=decimal_converter($RA);

return (@RA, @DEC);
}
```

This subroutine first calculates the declination and the Hour Angle. Then, using the current Local Sidereal Time, it finds the Right Ascension. Now the RA and DEC can be sent to the mount and the slewing procedure remains the same.

3.6.5 Alignment

Aligning the mount is the process of initializing the Gemini pointing software so it can accurately point to positions on the sky. The Gemini pointing model can also be refined through the alignment process. The alignment, or alignment model, of the mount is a model of the sky which the Gemini System uses to determine how to point the camera. The Gemini mount must be aligned with the sky by using the hand controller. This is not done automatically. But this need not be done automatically because when the mount is accurately aligned it should not need aligning again because it stores the alignment parameters in memory. Before aligning the mount, it must be given information on the observing site. It must know the longitude, latitude, local time and date, and the time zone of the observing location. These values can be sent to the mount with SLICAR.

Aligning the mount involves a tedious process of aiming the camera at an alignment star and then taking an image through the camera to check the position. The camera position can be adjusted with the hand controller, or it can be done through the computer with a PERL program named align_mount.pl (see Appendix C). This program allows the user to enter a variety of different slewing commands. The program also sets all the mount parameters. When the star is centered in the image the mount can be aligned. Doing this involves scrolling through the menu on the Gemini with the handcontroller to find the "Align Telescope" menu item. After choosing this, the user can choose "Initial Align" if this is the first alignment star or "Additional Align" for alignment on additional stars. Next the star can be found by scrolling through the list of bright stars and selecting the star chosen for alignment. This tells the Gemini system that it is pointing at that star. Now that the camera is aligned, additional alignment is recommended but not necessary. Additional alignment on other stars will increase the accuracy of the alignment. Up to 256 alignment stars are allowed in the pointing model. The accuracy of the alignment depends on the pointing precision while aligning on stars. Through the process of aligning the mount the Gemini software determines the accuracy of the polar alignment of the mount.

The alignment model will remain in the mount as long as it has power. If turned off the memory will be preserved until the internal battery is drained or removed. If there is a power loss, and the battery is operational, the mount will not need to be re-aligned. If the alignment parameters remain in memory, after powering up the Gemini the first thing the Gemini will do is prompt the user with the question "Warm Restart?". Here the user should press the rightmost button on the hand controller to select a "Warm Restart". By selecting this all the previous alignment information is kept. Caution must be taken not to inadvertently select "Cold Start" because all alignment information will be lost.

One peculiar aspect of the mount involves the alignment model. The Gemini system keeps track of time with an internal clock. Like any computer clock, this internal clock will eventually gain or lose time when compared with standard time. So to keep accurate time in the Gemini system it can be updated, and synchronized with the main computer clock (the computer updates its internal clock daily with access to the Internet). When the Gemini

clock is updated, the alignment model for the sky is not updated with it. Instead it acts like the model for the sky has a separate clock, which runs at the same pace as the Gemini internal clock, but does not get updated along with the internal clock. This can cause a lot of error if the Gemini clock is updated. In fact the error will be exactly the difference in time between the updated version of the time, and the incorrect time. This directly affects the right ascension coordinate because the local sidereal time (LST) is calculated from the internal clock.

Having realized this difficulty, SLICAR is made to compensate for this error. Instead of updating the internal clock of the Gemini Mount, SLICAR simply inputs the LST which the Gemini system is reporting. The program finds the difference between the Gemini LST and the correct LST (correct LST is based on the updated PC time). Next the right ascension of each field is amended based on this difference. The time difference (Gemini LST − Real LST) is subtracted from the real right ascension to obtain a corrected right ascension. The correction is calculated once per session, and applied to every coordinate for that session. The time difference is also recorded to the log file.

3.7 The Roof

3.7.1 The Roof as a Watch Dog

The observatory roof is a simple roll-off design. An electric motor, attached to the roof, is connected to a fixed chain. The chain spans a support structure which holds the roof while in the open position. When the motor is activated it crawls along the chain pulling the roof along with it. See figures 3.11(a) and 3.11(b) of the closed roof and figures 3.12(a) and 3.12(b) of the open roof. When the roof reaches the end of the support structure it triggers a limit switch which deactivates power to the roof. An identical limit switch is utilized when closing the roof. Each limit switch is a bumper. When the roof reaches the end of the track it bumps into the switch and deactivates the motor. A switch on a control box located inside the observatory control room lets the operator open and close the roof when needed.

Operation of the observatory roof is crucial because it provides protection to sensitive equipment when the weather turns bad. Therefore, precautions have been taken to ensure that the roof will perform reliably. This is done by employing another BASIC Stamp in the design of the roof. The Stamp included in the roof design is the key component which integrates the roof into the entire observatory design. This Stamp is also programmed to operate the roof independently should the need arise.

The roof Stamp performs four major functions. The first function is to accept commands from SLICAR to open and close the roof. The second function the Stamp performs is to monitor the progress of the roof while opening and closing. The third is to monitor SLICAR and the main computer. The last function is to detect power failures. By constantly keeping

Figure 3.10: The closed roof. The building seen on the left side of 3.11(b) contains the control room. The mountain in the background is Salt Pond Mountain in Giles County, VA

(a) View from under support structure.

(b) View from southwest of building.

Figure 3.11: The open roof. The building seen on the left side of 3.12(b) contains the control room.

(a) View from under the support structure.

(b) View from west-southwest of building.

an eye on these different systems the roof system can guard against danger. The roof system acts like a loyal watch dog, it protects the equipment from danger even if all other systems fail.

The roof Stamp is connected to the main computer through a standard serial connection. With this connection SLICAR is able to communicate with the roof Stamp. The roof Stamp

is programmed to accept and follow commands from SLICAR. The three possible commands are open the roof, close the roof, and check the roof status. The commands to open and close the roof are self-explanatory. The status check command is detailed below. When the roof Stamp receives an open or close command it immediately proceeds to open or close the roof. When the roof has finished opening or closing the Stamp then sends a message to SLICAR indicating that roof has finished the operation, then the system can proceed.

While the roof is opening or closing the Stamp is constantly monitoring the open or close limit switches. The Stamp knows when an open or close operation is complete when one of these switches is triggered. A malfunction in the roof mechanism can be detected in this way. The Stamp waits a specified amount of time for a limit switch to be triggered. If the limit switch is not reached by the roof within this amount of time the Stamp detects a roof malfunction and sends a message to SLICAR to act accordingly.

The situation in which the roof fails to close could potentially be very debilitating. For example, SLICAR might have commanded the roof to close because the cloud monitor detected approaching clouds. In this case, if the roof failed to close due to a malfunction, the components of the camera could become damaged from precipitation. Therefore, built into the roof Stamp is an emergency response system. If this actually happens then the roof Stamp is able to call the observatory administrators, via an electronic telephone dialer, to alert them of the problem. The electronic telephone dialer plays a pre-recorded message when the receiving end of the call is picked up.

The third major function of the roof Stamp is to monitor SLICAR. This is done through the status check command. SLICAR is designed to periodically check the status of the roof. As an example, the Stamp program will check the status of the roof and, if the roof is open, the Stamp tells SLICAR that the roof is open. The Stamp, when not opening or closing the roof, is always waiting for commands from SLICAR. The Stamp will only wait a certain amount of time for a command. After this time has passed without receiving communication from SLICAR the roof Stamp will automatically close the roof. This procedure is to protect against SLICAR or the computer freezing or crashing. The status check command is a way for SLICAR to tell Stamp that everything is still working correctly. It is also a way for the Stamp to tell SLICAR the current condition of the roof.

As mentioned above, the main computer and the roof system are both connected to a backup power source called an Uninterruptable Power Supply or UPS. In the event of power loss, the UPS does not store enough power to run the complete system through a night of observing. In the event of a power failure, the system can use the backup power to shut down gracefully. Since the roof must continue to operate, even when everything has failed it, the roof system watches for power failures. After this detection the Stamp immediately closes the roof. The next time SLICAR asks for the status, the roof Stamp replies by indicating a power failure. Then SLICAR can initiate the shut down procedure.

3.7.2 Operating the Roof

The Stamp used in the design of the roof is identical to the Stamp used for the focuser, filter, and dewar refill systems. The roof Stamp operates independent of the other systems to ensure that the roof will continue to operate should any other system fail. The PBASIC program loaded onto the roof Stamp communicates with SLICAR through a serial port. The roof Stamp also monitors the switches and detects power failures through the general I/O pins.

Serial communication with SLICAR is done in the same way as in the filter, focus, and dewar refill systems. SLICAR send commands to the roof Stamp through another serial port (this port is called $roof_port) like this:

```
$roof_port− >write($roof_command);
($count_in, $echo)=$roof_port− >read(1);
```

In these statements $roof_command will be equal to 1, 2, or 3. The Stamp interprets a 1 as a command to open the roof, and a 2 as a command to close the roof. When $roof_command is equal to 3, this indicates a status check.

The roof Stamp accepts commands with this PBASIC code:

```
SERIN 16,16780,100,Wait_command,[DEC1 command]
     IF command=1 then OPEN
     IF command=2 then CLOSE
     IF command=3 then CHECK_STATUS
```

Recall that the third argument in the `SERIN` statement instructs the program to wait for input from the serial port only for a set amount of time. If no data is received the program skips to the label given as the fourth argument. In the above example, after 100 milliseconds, if no data is received the program goes the line of code labeled `Wait_command:`. If information is received it is stored in the variable called `command`, and the Stamp will then proceed to check `command` with the three `IF` statements that follow the `SERIN` command.

The block of code above is contained within a loop in the PBASIC program. While SLICAR is operating other systems (not the roof) this loop is executed once every 100 milliseconds. This is because the `SERIN` command waits 100 milliseconds, stalling the loop. The label `Wait_command` comes prior to the input statement, so when there is no data inputted the program jumps up and repeats.

Each time through the loop the roof Stamp will perform a number of tasks (the PBASIC code for this loop can be found below). The first function is to check the manual switch.

A switch on the front of the roof control box allows a user to "manually" open the roof. A person can push the switch to open the roof without using the main computer. The switch was left in the design to allow the roof to be operated in the case of an emergency. The manual switch is wired to two input pins of the BASIC Stamp. P3 watches to see if the manual switch is switched to "open" and P4 watches to see if the manual switch is switched to "close". P3 is named `sw_open` and P4 is named `sw_close`.

The Stamp will also check for a power failure during each succession of the loop. A relay is connected to outside power, the wall outlet. This relay is also connected to P10 of the roof Stamp. P10 is named `check_power`. While the relay is open the state of P10 is low, or zero, and there is power to the building. If the power fails this relay will close, switching the state of P10 to high, so `check_power` equals 1. When this happens the Stamp will automatically close the roof.

The last operation performed in the loop is to decrement a counter, and check the value of the counter. The counter is a way to keep track of the amount of time that has passed since the last command was received from SLICAR. When the counter reaches zero, too much time has passed and the roof will close. The counter is initialized to allow approximately twenty minutes before timing out, and closing the roof. When a command is received the timer is reset. This simple procedure allows the roof system to monitor the status of SLICAR. If the computer should crash, or should SLICAR freeze, the roof Stamp will soon realize there is a problem and proceed to close.

The command loop described above is shown here:

```
Start:
     counter=time

Wait_command:
     roof_pwr=0
     roof_dir=0
     IF sw_close=0 THEN CLOSE
     IF sw_open=0 THEN OPEN

     IF check_power=1 then POWER_OUT
     counter=counter-1
     IF counter=0 then TIME_OUT

     SERIN 16,16780,100,Wait_command,[DEC1 command]
     IF command=1 then OPEN
     IF command=2 then CLOSE
     IF command=3 then CHECK_STATUS

GOTO Wait_command
```

The two lines which read `roof_pwr=0`, `roof_dir=0` are to ensure that power to the roof is deactivated. After a command is received and followed the program will go to the program label `Start:` where the time-out counter is reset with `counter=time`.

When a command is received the program will jump out of this loop at the `IF` statements. The program will go to the program label associated with the particular command. If SLICAR sends a 1 to the Stamp, the Stamp moves to the label `Open`. At the label `Open` the PBASIC program initiates the roof opening procedure.

The Stamp controls the opening and closing of the roof through four I/O pins. P1, named `roof_pwr`, is able to supply power to the motor which moves the roof. Setting `roof_pwr` equal to zero (0) deactivates the motor and setting `roof_pwr` equal to one (1) will activate the motor. P2, called `roof_dir` determines the direction the motor will run, open or close. To set the direction to open, `roof_dir` is set equal to zero (0), to close `roof_dir` is set equal to one (1). The last two pins used in controlling the roof are connected to the limit switches. The limit switch which stops the roof after it opens, called the open limit switch, is wired to P8. P8 is named `open_lim` in the PBASIC program. The `open_lim` pin is equal to zero when the roof is fully open. The limit switch which stops the roof after it closes, called the close limit switch, is wired to P9. P9 is named `close_lim` in the PBASIC program. The `close_lim` pin is equal to zero when the roof is fully closed.

The procedures for opening and closing the roof are very similar. When SLICAR intends to open (close) the roof it will send a 1 (2) to the Stamp. The Stamp will interpret this as an open (close) command, and proceed to the program label `Open:` (`Close:`). The first thing that happens is the direction bit is set to 0 for opening (1 for closing). Next, power is given to the roof by setting `roof_pwr` equal to 1. A status variable is set and a different counter is initialized.

Now the program enters another loop. This loop first checks the variable `open_lim` (`close_lim`). Next it checks the `sw_close` (`sw_open`) variable. The new counter is decremented and checked. Then the loop is restarted.

There are three situations which will kick the program out of this loop. The first is when `open_lim` (`close_lim`) is switched to zero. This indicates that the roof is fully open (closed) and the operation was successful. The second is when the manual switch is switched to close (open). Here the program will jump to the close (open) routine and close (open) the roof. The last situation is when the counter reaches zero. This means that the roof failed to open (close) in time, and there has been a malfunction.

When the roof has finished opening (closing), as indicated by the limit switch, the Stamp disables power to the roof, and immediately sends a reply to SLICAR which indicates the roof is open (closed). Finally the PBASIC program returns to the label `Start:`, resets the first counter, and enters the loop which waits for a command from SLICAR.

The opening procedure is coded as follows:

```
OPEN:
     roof_pwr=0
     roof_dir=0
     roof_pwr=1
     status=1
     counter2=time2
     OPENING
          IF open_lim=0 THEN HALT_OPEN
          IF sw_close=0 THEN WAITCLOSE
          PAUSE 100
          counter2=counter2-1
          IF counter2=0 THEN PHONE_HOME
       GOTO OPENING

HALT_OPEN:
     roof_pwr=0
     status=2
   GOTO SEND_STATUS

WAITCLOSE
     roof_pwr=0
     roof_dir=0
     status=3
     PAUSE 2000
   GOTO CLOSE
```

The PHONE_HOME and SEND_STATUS routines are described below. The closing procedure is very similar to the opening procedure.

The BASIC Stamp will send a message back to SLICAR after every command received, open roof, close roof, and check status. The send status statements in the PBASIC program are:

```
CHECK_STATUS:
     IF open_lim=0 THEN SEND_OPEN
     IF close_lim=0 THEN SEND_CLOSE
   GOTO SEND_STATUS

SEND_OPEN
     status=2
```

```
   GOTO SEND_STATUS

SEND_CLOSE
     status=4
   GOTO SEND_STATUS

SEND_STATUS
     SEROUT 16,16780,[DEC1 status]
   GOTO Start
```

The variable named `status` is a variable which stores the current status of the roof system. Through each step in the roof programming the status variable is reset to reflect the condition of the roof. The status variable is sent to SLICAR at the `SEND_STATUS` label. The main computer checks this information and acts accordingly (this information is detailed below). The `SEND_OPEN` routine, called after the opening procedure, sets the status variable equal to open then calls `SEND_STATUS`. The `SEND_CLOSE` routine, called after the closing procedure, sets the status variable equal to close then calls `SEND_STATUS`.

When the command sent to the Stamp is 3, a status check, the `CHECK_STATUS:` routine is implemented. Here the program checks the state of the limit switches to see if the roof is open or closed. If the roof is neither open nor closed the last value of `status` is sent to SLICAR.

In the command loop illustrated above there is a block of statements which reads:

```
IF check_power=1 then POWER_OUT
counter=counter-1
IF counter=0 then TIME_OUT
```

In the case where the power goes out, the variable `check_power` will equal 1 and the program will go to the label `POWER_OUT`. The `POWER_OUT` routine sets the `status` variable and calls the label `EMERGENCY_CLOSE:`. If the timing counter reaches zero then the program moves to `TIME_OUT`. `TIME_OUT` also sets the `status` variable and calls the label `EMERGENCY_CLOSE:`.

The `EMERGENCY_CLOSE:` section of the roof program is identical to the close routine except for one thing. The `EMERGENCY_CLOSE:` routine includes input and output commands. Within the `EMERGENCY_CLOSE:` loop are these commands:

```
SERIN 16, 16780,100,EMERG_CLOSE_LOOP,[DEC1 command]
SEROUT 16,16780,[DEC1 status]
```

These two lines serve the purpose of checking for communication from SLICAR. After a time-out or a power outage, it is possible that SLICAR could recover and send a command to the roof system. These statements are available to receive that command and send the current status back to SLICAR. The status in this case will tell SLICAR that there was a power failure or the roof Stamp timed out. The commands looped over in `EMERG_CLOSE_HALT:` serve the same purpose.

The purpose of the counter inside the opening and closing routines is simple. This counter times the progress of the roof as it is opening or closing. The limit of this counter is set to five minutes. If, after 5 minutes, the roof has failed to open or close the program skips to the routine labeled `PHONE_HOME`. The purpose of `PHONE_HOME` is to alert an observatory administrator of the malfunction.

The roof Stamp is connected to an automatic telephone dialer through P11, named `auto_dial`. The phone dialer is preprogrammed with phone numbers of the observers. An emergency message is recorded on the unit. When P11 is set equal to 1 the phone dialer is activated. The Stamp will continue to enable the phone dialer to call periodically until the roof unit is turned off or reset. This insures that an operator will respond and report to the observatory.

In any case where the roof system implements an emergency procedure, timing out, power outage, or phoning home, the Stamp can not return to regular operation without first being reset. This feature is a result of the PBASIC program running on the roof Stamp. Therefore, after an automatic shut down is prompted by the roof system, the problem which caused the shut down should first be remedied. To continue observations the Stamp controlling the roof *must* be reset. Resetting the roof Stamp is done by unplugging the roof control box, and plugging it back in.

The entire PBASIC program for the roof Stamp, including documentation, can be found in Appendix E.

3.7.3 Roof Control with SLICAR

As illustrated above, the Stamp operating the roof functions independent of SLICAR. Because of this design SLICAR must be able to determine the status of the roof. The status of the roof is sent to SLICAR whenever SLICAR sends a command to the roof Stamp. Even in the cases where the Stamp closes because of a malfunction or emergency, if SLICAR sends a command to the Stamp, the Stamp will reply with the status of the roof system. The Stamp always keeps track of the state of the roof by always setting a variable called `status`. For every possible situation the variable called `status` has a unique value which SLICAR recognizes. The possible values of `status`, and the description of what each value represents is given in table 3.5.

After SLICAR sends a command to the roof Stamp it calls a subroutine named roof_status. This subroutine receives the response from the Stamp and interprets the response with a

Table 3.5: Roof Status Table.

Status Value	Description
1	The roof is opening.
2	The roof is open.
3	The roof is closing.
4	The roof is closed
5	The roof Stamp timed-out!
6	The roof is in transit.
7	Power outage. Roof automatically closed.
8	Roof failed to open/close. Phone Home!

series of if statements. After the status of the roof is determined SLICAR will do one of two things. It will either continue with the observing procedure or shut down the system. SLICAR will shut down for conditions 5, 7, and 8 in table 3.5. As mentioned in the previous section, the roof Stamp needs to be reset before resuming the survey if any of these situations occur.

3.8 The Cloud Monitor

When the sky becomes cloudy, observing must be suspended. Not only is there no useful information gained from observing clouds, but the instrument must be protected if it happens to rain. Incorporated into the automated system is a cloud monitoring device.

The cloud monitoring system, designed by Dr. John Simonetti, is a CCD camera which is placed outdoors and directed at Polaris. Figure 3.12 is a picture of the Polaris camera inside a weather-proof housing.

The camera system periodically records an image of the sky and finds the brightest point on the image. The brightest point, which is Polaris, is measured against the background brightness of the surrounding area. This value for the peak is downloaded to the main computer and recorded in a text file. The peak value can then be looked at and processed by SLICAR to determine the current viewing conditions.

A separate PERL program, written by Dr. John Simonetti and Phillip Nelson and running in parallel with SLICAR, controls and monitors the Polaris camera system. It sends commands to the camera to take images, and it monitors the value of the peak intensity. Each time the value of the peak is received it is added into a sliding average of the last five values of the peak. The peak and average intensities, along with some other useful information, are included in a separate log file which is updated every time an image of Polaris is taken.

Figure 3.12: Two views of the cloud monitoring camera mounted on the side of the control building.

(a) (b)

When the running average drops below about $\frac{1}{3}$ of the peak value then the sky is assumed to be clouded over.

SLICAR will check the cloud monitor log file at the beginning of the observing session, and after each image taken. A subroutine called &check_polaris is responsible for checking the cloud monitor file. This subroutine is as follows:

```
sub check_polaris
{
    $polaris_file = 'C: \ SLIC\\ DATA\\'.$date_name.'\\'.$date_name.'.POL\';
    if (-s \$polaris_file) {
        open (POLARIS, $polaris_file);
        $. = 0;
        do {$line= <POLARIS> } until $. == desired_line_number | eof;
        chomp $line;
        seek(POLARIS, 0, 1);
        close (POLARIS)}
    else {
        &tasks_done("UNABLE TO LOCATE POLARIS MONITOR FILE, $polaris_file.");
        &shut_down}

    $intensity= substr($line, 76, 8);
    close (POLARIS);
    if ($intensity <= 2500.00){
        &tasks_done("* * * WEATHER CONDITIONS PROHIBIT OBSERVING. * * *");
        &shut_down}
}
```

This subroutine, coded by Phillip Nelson, reads the last line in the Polaris monitor log file and stores each piece of information in an array. Then it singles out the value of the average of the last five peak intensities. The routine then compares this value with 2500.00 (this value is approximately $\frac{1}{3}$ of the typical Polaris peak intensity). If the average peak value falls below this number then SLICAR determines that conditions are not favorable for observing because it has become cloudy. Then observing is halted, and SLICAR shuts the system down.

The program which runs the Polaris monitoring camera can be found in Appendix B.

3.9 Camera Control

Images from the CCD camera are exposed and processed through the electronics designed by Photometrics, and contained with the CE200A Camera Electronics Unit. The user can interface with the camera unit through the PMIS Image Processing Software. PMIS is Microsoft© Windows compatible software which allows control of the camera, and processing of images. Image acquisition and processing can be done by entering commands in the PMIS command window. SLICAR is able to control the camera by sending these commands to the PMIS software.

For each image there is a standard process for acquisition. First the CCD is exposed for a specific time period. Next PMIS downloads the image from the CE200A Camera Electronics Unit to the PC. The image is then converted and saved in the image directory as a FITS image. Some of these operations are performed by invoking a set of macros in PMIS.

The initialization process is completed in a subroutine of SLICAR called PMIS_start. To initialize communication with the camera software this command is executed by SLICAR:

```
$Client=new Win32::DDE::Client('PMIS', 'CLI');
```

This command runs a Dynamic Data Exchange (DDE) package designed for PERL. This also sets the DDE client as the PMIS software. Now any command sent to $Client will be written to the PMIS command line. An example PERL command to send data to the PMIS command line follows:

```
$Client− >Execute(command);
```

Where command is the PMIS command to enter on the command line.

Before using PMIS to acquire images some settings must be initialized. First some information is given to PMIS. The following commands define some values that will be included in the saved image file.

```
$Client− >Execute("vdefine iiii \"$directory\"");
$Client− >Execute("vdefine lens $lens");
$Client− >Execute("vdefine ctemp $ctemp");
$Client− >Execute('!c:\slic\newmacs\startnew.cmd');
sleep(2);
$Client− >Execute('menu newimage');
sleep(2);
$Client− >Execute("OBS 1");
```

The statement vdefine is used in the PMIS software to define a variable. The first statement above tells the software the directory to which images are saved. The next two lines give the type of lens used (58mm) and the camera temperature (-110.5) respectively. The fourth statement executes a macro called startnew.cmd which further primes PMIS for observing. This macro can be found in Appendix F. The statement $Client− >Execute('menu newimage'); opens a new image window in the software. The last line records an image for one millisecond. This first image acquisition is necessary because the first image recorded after opening PMIS is usually worthless, so this command clears this problem. Now the camera software is initialized and ready for observing.

The process of taking and saving an image involves several steps. First, the exposure time must be known. Next, a waiting time is calculated from the exposure time. The wait time is equal to the exposure time plus twenty seconds. This waiting time is the time that SLICAR will wait for the image to be exposed and processed. SLICAR must wait because there is no way for PMIS to send information to SLICAR, for example, to tell SLICAR when the exposure is complete. Now the command to acquire the image can be sent to the PMIS software:

```
$Client− >Execute("OBS $exposure_time");
sleep($wait_time);
```

The command OBS is the PMIS command which tells the camera to expose the CCD. The variable $exposure_time is set to the number of milliseconds the exposure should last. The statement sleep($wait_time); makes SLICAR wait for $wait_time seconds, where $wait_time is equal to $exposure_time plus 20 seconds. After this wait time the image will be acquired and downloaded to the PC. Now the image can be processed and saved to disc. The SLICAR subroutine which performs these tasks is called save_image. First some variables in PMIS are set:

```
$Client− >Execute("vdefine object \"$field\"");
$Client− >Execute("vdefine ifilter $filter");
$Client− >Execute("vdefine type \"$image_type\"");
$Client− >Execute("vdefine fname \"$file_name\"");
```

This information is used as documentation in the final FITS image. Where object, ifilter, and type are the names of the object in the image, the filter used, and the type of image(such as bias, or flat) respectively. The value fname is the file name under which the image will be saved. Next the subroutine invokes this statement:

```
$Client− >Execute('!c:\slic\newmacs\fitsnew.cmd');
```

This command executes a predefined macro which first converts the image into a FITS file, and then saves that image as the file named fname.fit. The images are saved in a directory which is named after the current date. The complete set of macros for PMIS can be found in Appendices F and G.

Chapter 4

Running and Maintaining the Automated Observatory

4.1 Starting the System

4.1.1 The First Step: The Observing File

Each observing session begins by building an observation instruction file, called the Observing File. The Observing File is a text file which contains all the information about the particular fields that will be observed. The Observing File is made by the observer and loaded onto the main computer. With algorithms written by Phillip Nelson, SLICAR reads this file and determines what fields to observe, and how to observe them.

The Observing File contains information about each field that will be observed for a particular session. It contains the observer's name, the RA and DEC of each field, the name of each field, the filter needed for each field, the exposure time for each image, and the number of images of each field. The Observing File can also include some commands to give to SLICAR. These commands are detailed later.

An example Observing File looks like this:

Observer Name
180000 404000 Her16 1 5 5
182037 −032559 Ser01 4 5 5
184952 471405 Lyr00 2 5 5
191854 404525 Lyr03 4 5 5
194637 -093123 Aql14 3 5 5
1205148 -071612 Aqr01 1 5 5

The first entry on a line in the Observing File is the right ascension of the field, the second is the declination. The right ascension is in the format HHMMSS, and the declination is in the format ±ddmmss. The third entry gives the name of each selected field. This name can be any 5 character name, but these names follow a standard naming convention. The entry following the name gives the filter number which is to be used for the field. The last two entries are the exposure time in minutes and the number of exposures for the field. SLICAR will start at the first line and read in the data. It will try to observe the field on that line by slewing to its coordinates, and changing the focus and filter. Then it will expose the CCD. When that is finished the next line is read. If two consecutive entries are of the same field (identical RA and DEC coordinates) SLICAR will not slew the mount again.

Some commands can be given in the Observing File. As an example, SLICAR can be forced to refill the dewar after a certain field. Normally SLICAR will refill the camera dewar after about three hours. This can be altered by inserting a separate line containing the word "FILL" in the Observing File. When SLICAR reaches the line with the word "FILL" the camera dewar refill procedure will begin. At the end of the dewar refill procedure SLICAR will continue where it left off in the Observing File. Another command that can be inserted into the Observing File is "FLAT". If the word "FLAT" is added into the file directly after the observer name, the first task SLICAR will perform is that of taking flat field images (more explanation of the flat fields images can be found in section 4.2.5). The last command that can be inserted is "EAST". If this is included at the end of a line containing information for a field then SLICAR will observe that field while it is east of the meridian (this is explained in more detail below).

The Observing File is named for the date of the observing session. The name is given in the format YY_MM_DD_OBS.txt and is saved as a text file on the main computer. For a given date of observing SLICAR will look for the Observing File for that date to start observations. If the file does not exist the session will not start.

An Observing File can be made by simply typing the information for each required field and saving the document. But it can also be made automatically. A PERL program, called simulate_vx.pl (vx refers to the version number), has been written by Phillip Nelson to build the Observing File. This program looks at the list of all the fields in the entire survey which is contained in another text file called survey.txt. This file contains all the required information for each field in the survey. It contains the coordinates, name, whether or not that field has been observed, and the filter with which the field has been observed, if any. Therefore the simulation program knows which fields have been recorded and which fields need to be completed. Based on the date and time, it chooses fields which will appear near the meridian for that session. It is preferential to observe near the meridian because this direction offers the best viewing conditions. All fields will be acquired between the hour angles of -15 minutes and 2 hours. These limits also prevent the mount and camera assembly from slewing to awkward positions. Also, fields are chosen that are above approximately 30° altitude. Anything below this will be obstructed by the observatory wall. Next, the program simulates observing, starting with fields that are near the meridian earliest in the session.

The program performs a mock observing session by advancing a clock through each process to simulate the real observing session. After each field, the program building the Observing File finds the next field which is closest to the meridian. Each field is added to the Observing File in succession.

In the cases where the observer wishes to observe a field that is not on the list, the Observing File can be edited simply by opening the file and adding the required information about the field. If the added field is not within the hour angle window, between -15 minutes and 2 hours, the word "EAST"can be added to the end of the line to tell SLICAR to ignore the hour angle limits.

At the end of each observing session it is important for the observer to review the images recorded and note the fields that have been successfully acquired in order to update survey.txt.

4.1.2 Start of the Program

After the Observing File has been created and saved on the main computer SLICAR can be started. If the devices to the observatory are not on, they must be turned on. This includes the focuser, filter wheel and dewar valve system, the Polaris monitor, the mount, and the roof. If observing has been ongoing, and error free, these system should already be activated. Some systems need to be checked. The resupply dewar should be checked to make sure enough LN_2 is available for the session. The focusing mechanism should be checked to insure the worm and camera gear are meshed correctly.

Next, the Polaris monitoring program can be started by clicking on its program icon. Now SLICAR can be started by clicking on the SLICAR icon. Once SLICAR is started all systems are configured. The serial ports are initialized for communication, the PMIS software is started, the Observing File is read, the mount is initialized, and data about the observing site is calculated. Now all systems are ready to go. This initialization procedure is implemented by calling a set of subroutines at the start of SLICAR:

```
use Win32::SerialPort qw( :STAT 0.19 );
use Win32::DDE::Client;
use File::Copy;

&configure_ports;
&initialize_mount;
&create;
&PMIS_start;
&site_data;
&input;
```

4.2 Observing Procedure

A simplified flow diagram of SLICAR can be seen in figure 2.1 in section 2.2. What follows is a detailed illustration of each step carried out by SLICAR .

4.2.1 Starting and Ending times

The starting and ending times of each observation session are based on two things, the Sun and the Moon. All of these observations will take place when the Sun and Moon are not visible. Therefore, SLICAR must know the times when the Sun and Moon are below the horizon. This is done by using tables provided by the United States Naval Observatory Astronomical Data Department. The first table gives the Moon rise and Moon set times for each day. The second table used gives the start and end times for astronomical twilight. These tables can be produced at: *http://aa.usno.navy.mil/data/docs/RS_OneYear.html*[23], the U.S. Naval Observatory website. They are saved on the main computer as twilight.htm and moon.htm and updated annually. A sample of each table can be found in the Appendix J and K. Each day the starting time and ending time for the observing session is calculated based on the astronomical twilight and moon tables. Phillip Nelson wrote the subroutine in SLICAR which reads from these tables to determine the starting and ending times for each observing session.

Observing always starts after evening astronomical twilight has ended, and always ends before morning twilight begins. Observations can not proceed when the Moon is visible (all phases), so if the Moon is above the horizon before the end of twilight, the start time is delayed until the Moon has set. Subsequently, if the Moon rises after evening twilight, the observing session will end prior to Moon rise. In cases at or near full Moon the length of the observing window will be very short, too short to make any meaningful observations. So if the observing time is less than 1 hour, that session is terminated. The subroutine in SLICAR called `starttime` reads from the tables and determines the start and end time for the observing session.

Once the start time is determined SLICAR will wait until that time. While waiting SLICAR is looping through a routine called `wait_to_start`. Each time through this routine SLICAR checks the current CPU time with the starting time. Also in this loop is a call to the roof subroutine `&roof(3);`. Recall from section 3.7, the argument given to the roof subroutine is a command for the roof Stamp to follow, in this case `3` tells the roof to send the current roof status. This command is sent to the roof Stamp approximately every 10 seconds to reassure the Stamp that SLICAR is still running effectively. If this command was not included in this waiting system then the roof Stamp would time-out after 20 minutes, causing the roof to close until the system is reset (see section 3.7).

4.2.2 Getting Ready to Observe

The observing procedure begins prior to the actual start of observations by filling the camera dewar and allowing the CCD to cool down to the operating temperature. The time when the dewar is filled is 2 hours prior to the start time. This permits ample time to allow the CCD chip to cool. The subroutine named &cool_down is called to allow the CCD time to cool. While the camera is cooling the status of the roof is checked periodically to prevent the Stamp from timing out. As the dewar cools, a significant amount of LN_2 boils off, leaving the dewar only partially full. So the camera dewar is topped off with LN_2 after the CCD has cooled.

The next step before observing is to check the state of the Polaris monitor. If the Polaris monitor indicates that the sky is cloudy then the session is terminated. When the sky is clear, observing can start.

After the Polaris check it is time to expose bias images. The bias images measure how much each pixel of the CCD is biased above zero. Bias images are exposed by taking an image with zero exposure time in complete darkness. The bias images are acquired through PMIS in the same way all images are acquired. A subroutine called &take_bias performs the procedure of recording 10 bias images, and saves them in the correct image directory for the current session. Originally the bias images were scheduled to be acquired at the end of the observing session. The procedure was changed because malfunctions in certain systems would cause the bias images for that night to be missed, or SLICAR may not have ever arrived at the bias procedure when a malfunction occurred. So the bias procedure was moved to the beginning of the session to avoid that potential problem.

At this point the roof is opened by the subroutine &roof(1) (In this case 1 tells the roof to open). Once the roof is open observations can begin.

4.2.3 Observing

The subroutine &observe in SLICAR implements the entire observing procedure by reading the input file and calling on a number of other subroutines which perform certain tasks. This routine will continue to loop through each line of the Observing File until the end of the file is reached, or until observations are terminated by another factor.

The observing routine executes a loop which follows a specific algorithm. First, a field is chosen from the Observing File. Next, SLICAR checks the current time to see if the dewar will need to be refilled. When it has been 3 hours or more since the last camera dewar fill, the refill subroutine is called to fill the dewar with LN_2. A "FILL"command in the Observing File will also cause the dewar to be refilled at this time. The next step in the observing loop is to check the current time against the quitting time determined when SLICAR was started. If the current time exceeds the quitting time the observations are halted.

Now the altitude of the selected field is checked. If the altitude is below 30° (the wall of the building) it can not be observed. So the next field in the observing list is found. When the altitude is in the viewable range the field can be imaged.

The first step in imaging is to slew the camera to point at the selected field. So the subroutine &slew(@coordinates) is called (recall @coordinates is an array containing the RA and DEC of the field). The first step in the slew procedure is to check the roof status. Again this is to prevent the roof Stamp from timing out. If there are no errors detected, the mount will slew the camera to the coordinates and automatically track that position. When the slew is finished SLICAR can continue.

After slewing the camera, the filter wheel is rotated to select the correct filter that was given in the Observing File. Now the filter wheel subroutine, &filter, is called. Again, the first step in this routine is to check the status of the roof. Then, SLICAR directs the Stamp to select the filter. Since each filter has associated with it a specific value for the focus, the filter routine calls the focusing routine with information regarding the filter that is in place. An image can be acquired after the focusing system focuses the lens correctly.

The image is acquired and saved through PMIS. The procedure will loop through this for each exposure of a particular field before it reads the next field information from the Observing File. For example, if five exposures of a field are required, the routine will loop five times before it moves on to the next field. Each time through this loop the procedure is repeated. The filter wheel is instructed to choose the correct filter, and the lens is focused. Normally these redundant instructions cause nothing to happen. They are included in the loop as an added check that each system is still operating properly. Also, each instruction includes a check of the roof system. The mount is not instructed to slew again because each image must overlap exactly.

When the end of the file is reached the observe routine is exited and SLICAR will shut the system down. Cloudy weather, power loss, and the arrival of the end time are other situations that cause SLICAR to terminate the session. SLICAR will also abort the observing session if an error is detected in any system, as detailed in chapter 3.

4.2.4 Shutting Down

When the observing is complete, or when SLICAR is aborted due to a device failure, SLICAR will shut down the system. The shut down procedure is contained in the subroutine &shut_down. This routine first returns devices to their storage positions where applicable. To shut the system down, the roof is closed, the focusing mechanism is returned to position zero (camera focused to infinity), the mount is slewed to the counterweight-down position and the tracking is turned off. All files are copied to a backup directory. Then SLICAR enters a loop which waits for the next night of observing.

In the case where SLICAR is aborted due to device failure the shut down procedure must

be altered. When a device fails the shut down procedure is automatically initiated. The shut down procedure then tries to store each device in the home position. When SLICAR tries to store the failed device the malfunction will be detected again, and the shut down procedure will again be started, causing an infinite loop. Built into each device controlling subroutine is an error flag to avert this disaster. When a malfunction is detected by SLICAR it raises the error flag, then shuts down. The shut down procedure looks for these flags. When the flag is detected for a particular device, the shut down command for that device is ignored. For example, the focusing mechanism will detect an error if the Stamp controlling the focuser does not respond within 10 minutes after being commanded to changed the focus. At this time the statement $focus_error = 'error'; is executed, which raises the error flag. Normally $focus_error = 'none', which means all is well with the focuser. Then the &shut_down subroutine is called when an error is detected. In the &shut_down subroutine there is a block of code which reads:

```
if ($focus_error ne "error")
    {&focus("0000");}
```

Therefore, when the $focus_error flag is not raised the focusing routine will command the camera focus back to zero. When the error flag is raised, this block is skipped. Similar algorithms are used for the mount system and the roof system.

When the observing session is completed without any errors or malfunctions all systems will be stored, and SLICAR will enter the loop waiting for the next session. This loop simply checks the current time, when the current time reaches noon, SLICAR restarts itself from the beginning. If the Observing File already exists for that date the automated system will continue to observe. The only maintenance required is to periodically replace the resupply dewar with a full one, update the field list, and build new Observing Files for each night. All image files and log files are saved on the main computer and can be retrieved at any time for further processing. All information is written to a Log File for the observer to read at the end of the session. The Log File is also named after the date, the format is YY_MM_DD.log.

4.2.5 Flat Field Images

The process of recording flat fields requires human presence. The inconvenience of taking flat fields is negligible because regular visits to the observatory are required as the LN_2 dewar needs periodic replacements. Also the flat field process is mostly automated and does not require a lot of time. At most, the flat field images are recorded on a weekly basis.

The process of taking the flats starts by entering the word "FLAT"into the Observing File. It must be entered on the line immediately after the name of the observer, and must be on the line alone. After SLICAR finishes acquiring the bias images (section 4.2.2) the flat fields can be imaged and saved.

The flat fields are used as a way of correcting images for variations in sensitivity across the image. The pixels on the CCD are not identical in that they have different sensitivities. Also there is a more general variation in sensitivity across the CCD. Pixels that are off axis have a smaller collection area because they are not perpendicular to the incoming light. So a flat field is set up and imaged by the camera to offset these types of variations. There are two flat fields. Each is produced using a large box with a translucent cover. Inside the box is a number of light bulbs. In the white, or continuum, light box the bulbs are ordinary incandescent bulbs. When the box is turned on it provides a flat field to be imaged with the continuum filter, the Hα filter, and the [SII] filter. The second box contains hydrogen tubes which, when turned on, provide a flat field of the hydrogen spectrum to be imaged through the Hα filter.

When the flat process is initiated the user is prompted by SLICAR to turn on the Hα light box, and press return when done. The program slews the camera so it is directed at the Hα box and proceeds to record the Hα flats. When this is finished, SLICAR prompts the user to turn off the Hα light box and turn on the white light box, and press enter when done. Now the camera is slewed until it is facing the white light box. The remaining flat fields are now recorded. After this is done the user will again be prompted, this time to turn off all lights in the observatory, lock the door, and press enter when done. Now the observing procedure will start up as usual.

All flats are saved in the directory of the current session with appropriate names.

4.2.6 Malfunctions and the Log File

For each task that SLICAR attempts, the task, the outcome of the attempt, and time of the attempt is recorded in a text file. Phillip Nelson designed coded the process of recording data to the log file. This log file is a record of all events which transpire throughout the observing session. This file is named after the current date, as YY_MM_DD.log, and saved in the directory of the current session. Along with being an accurate record of the survey, the log file can be reviewed to evaluate the progress of the observatory and identify any problems.

The first step in troubleshooting a problem is too review the log file. When the observatory shuts down due to failure of a device SLICAR will make a note in the log file of the device which failed and the reason why the failure was detected. In most cases the problem will reveal itself in this record. An example log file can be found in Appendix I.

Chapter 5

Conclusions

The Virginia Tech Spectral-Line Survey is ideally suited for an automated system. The observatory was successfully automated by tackling the problems presented by each component individually, then meshing each piece together to complete the final system. A combination of mechanical, electronic, and software solutions have been utilized to yield a more efficient observatory.

The automation process has brought to light many potential problems, and measures have been taken to try to prevent every problem encountered. Various routines which monitor different devices are built into the system to detect failures. The first step in solving a problem is to recognize that it exists. Although every precaution has been taken to eliminate malfunctions, it is impossible to test or predict every possible situation for such a complex system. Because of this, the overall design of the automated system is flexible and adaptable. The main programming and procedures can be altered to accommodate unforeseen complications.

The SLIC observatory is capable of functioning on its own for several days at a time. All it requires is a regular updated Observing File. Updates to the Observing File, and even to SLICAR itself, can be done remotely through a network connection. After several days the LN_2 resupply dewar needs to be refilled or replaced, and once in a while an observer assists the system when acquiring flat images. These tasks comprise a minimal amount of attention to the observatory. It is possible to upgrade these systems to have the observatory entirely automated. Turning light boxes on and off is a simple task, and that might be automated in the near future. It is also conceivable to allow SLICAR to build its own Observing File each night. On the other hand, an autonomous LN_2 replenishing system is much larger project. Constructing a program that will automatically determine the correct focus is also not a small task, but possible.

Although it is designed primarily to conduct the Virginia Tech Spectral-Line Survey, the SLIC automated observatory can be utilized for a variety of other projects, and can be a

valuable educational tool as well. The versatility of this instrument insures a long future of research and discovery.

Bibliography

[1] J. Bland-Hawthorn and R. Reynolds. Gas in Galaxies. *To appear in Encyclopedia of Astronomy & Astrophysics*, 2000.

[2] L.J. Boyd, R.M. Genet, and D.S. Hall. Automatic Telescopes Large and Small. *Publications of the Astronomical Society of the Pacific*, 98:618–621, 1986.

[3] B. Dennison, J.H. Simonetti, and G.A. Topasna. An Imaging Survey of Northern Galactic Hα Emission with Arcminute Resolution. *Publications of the Astronomical Society of Australia*, 15:147–148, 1998.

[4] B. Dennison, G.A. Topasna, and J.H. Simonetti. Detecion in Hα of a Supershell Associated with W4. *Astrophysical Journal*, 474:L31–L34, 1997.

[5] Katia M. Ferriere. The Interstellar Environment of Our Galaxy. *Reviews of Modern Physics*, 73:1031–1066, 2001.

[6] A.V. Filippenko. Technical Description of the Berkeley Automatic Imaging Telescope. *Publications of the Astronomical Society of the Pacific, ASP Conference Series*, 34:115–122, 1992.

[7] A.V. Filippenko. The Scientific Potential of Automatic CCD Imaging Telescopes. *Publications of the Astronomical Society of the Pacific, ASP Conference Series*, 34:55–66, 1992.

[8] A.V. Filippenko. The Wisconsin APT, The First Robotic Telescope. *Publications of the Astronomical Society of the Pacific, ASP Conference Series*, 34:3–8, 1992.

[9] R.M. Genet, L.J. Boyd, and D.S. Hall. Variable Star Observations with Automatic Telescopes. *International Astronomical union*, 118:47–55, 1986.

[10] R.K. Honeycutt, B.R. Adams, D.J. Swearingen, and W.R. Kopp. Devices for Observatory Automation. *Publications of the Astronomical Society of the Pacific*, 106:670–674, 1994.

[11] R.K. Honeycutt and J.E. Kephart. Some Applications of Microcomputers in Observatory Automation. *Publications of the Astronomical Society of the Pacific*, 94:605–609, 1982.

[12] R.K. Honeycutt and G.W. Turner. Architecture of the Software for the Indiana CCD Automated Telescope. *Publications of the Astronomical Society of the Pacific*, 94:77–78, 1992.

[13] J.F. McNall, T.L. Miedaner, and A.D. Code. A Computer Controlled Photometric Telescope. *The Astrophysical Journal*, 73:756–761, 1968.

[14] A.H. Mintner, R.J. Reynolds, B.D. Savage, J.M. Cordes, R.J. Reynolds, and J.M. Shull. Conference Highlights: Green Bank Workshop on Warm Ionized Gas in Galaxies. *Green Bank Workshop*, 112:424–426, 2000.

[15] R.J. Reynolds. Ionizing the Galaxy. *Science*, 227:1446–1447, 1997.

[16] R.J. Reynolds. The Gas Between the Stars. *Scientific American*, 286:34–39, 2002.

[17] R.J. Reynolds, S.L Tufte, L.M. Haffner, K. Jaenig, and J.W. Percival. The Wisconsin Hα Mapper (WHAM) : A Breif Review of Performance Characteristics and Early Scientific Results. *Publications of the Astronomical Society of Australia*, 15:14–18, 1998.

[18] M.W. Richmond, R.R. Treffers, and A.V. Filippenko. The Berkely Automatic Imaging Telescope. *Publications of the Astronomical Society of the Pacific*, 105:1164–1174, 1993.

[19] J.H. Simonetti, B. Dennison, and G.A. Topasna. The Contribution of Galactic Free-Free Emission to Anisotropies in the Cosmic Microwave Background Found by the Saskatoon Experiment. *The Astrophysical Journal*, 458:L1–L3, 1996.

[20] R.R. Treffers, M.W Richmond, and A.V. Filippenko. Technical Description of the Berkeley Automatic Imaging Telescope. *Publications of the Astronomical Society of the Pacific, ASP Conference Series*, 34:115–122, 1992.

[21] Gemini Command Description webpage. http://www.docgoerlich.de/l2v1serial.html.

[22] Losmandy Astronomical Products Website. http://www.losmandy.com.

[23] US Naval Observatory Website. http://aa.usno.navy.mil/data/docs/rs_oneyear.html.

Appendix A

The Main PERL Program: SLICAR.pl

This is the source code for the main program SLICAR (Spectral-Line Imaging Camera Automation Routine).

```
# # # # # # # # # # # # # # # # # # SLICAR.pl # # # # # # # # # # # # # # # # # #
#        (Spectral Line Imaging Camera Autmation Routine}
#
#!/usr/bin/perl
#
#CREATED BY: Phillip Nelson and Ken Portock
#DATE: MONDAY APRIL 15, 2002
#UPDATED: WEDNESDAY, DECEMBER 11, 2002
#
#This is the main control program for the SLIC camera


#------ MAIN PROGRAM -------------------------------------------------------------#

system 'cls';  #DOS cls (clear screen) command

use Win32::SerialPort qw( :STAT 0.19 );  #Open module for Stamp communication
use Win32::DDE::Client; #opens DDE module library to communicate w/ PMIS
use File::Copy;  #Load PERL 'Copy' module to use copy(old_loc, new_loc) cmd.

&configure_ports; # Information for communicating through the serial ports
&site_data;             # Get information about SLIC location and a few constants
&create; # Create directory and log file for this session
&PMIS_start; # Initializes PMIS, PMIS must already be open, directory must exist
&input;  # read input file
&initialize_mount; # Initialize Gemini mount settings
&initialize_focus; # Reads previous focus position
&startime; # Read twilight and moonrise/-set tables to determine start and end
times
#&wait_to_start; # Loop until the start time indicated by startime
&header_info; # Prints some preliminary info to logfile
&dewar;  # Initial fill of dewar
&cool_down; # Wait 2.0 h for camera to cool to -110.5 C
&check_polaris;  # Check to see if the sky is clear/cloudy
# If yes, observe. If not, shut down.
&dewar;  # Refill dewar to replace evapoarted nitrogen
&take_bias; # Take bias images at beginning of observing session

#&flat_fields_main if ($inputdata[1] == "FLAT"); #Take flat fields?
$roof_status_variable = 0;   #print roof status message once
&roof(1);      # Open roof (See 'sub roof' for details)
&observe;     # Observes fields listed in OBS.txt
&shut_down;     # Power down and go to stand-by.

#------ END MAIN PROGRAM ---------------------------------------------------------#


#------ SUBROUTINES --------------------------------------------------------------#

sub altitude { ##########################################################################
#Calculate the altitude of a field and determine whether it is visible from our
#location.  Computations for determining $ALT and $AZ were adapted from the book,
#_Practical Astronomy With Your Calculator, 3rd Edition_, by Peter Duffett-Smith,
#pgs. 35 & 36
#
#1. How many days since January 0.0, 2000? (@UT is an array that gives
#the current Universal Time when this program runs. 'gmtime' is a built-
#in PERL function that reads the system clock and returns GMT. The
#array elements are identical to the elements of the array given by the
#'localtime' funtion used in 'sub create.')

#1. Calculate UT
```

```
  @UT_now = gmtime;   # Current UT
  $UT = DECIMAL(1.0, $UT[2], $UT[1], $UT[0]);  # UT in hours
  $LST = UT2LST($UT, $UT[3], $UT[4] + 1, $UT[5] + 1900);  # LST in degs.

#2. Calculate Local Sidereal Time given UT, # days since J2000.0 and Longitude
  $LST = UT2LST($gem_UT, $gem_dd, $gem_mn, $gem_yr);    #Gives LST in hours
  $LST = 15.0*$LST; #Gives LST in degrees

#3. Compute hour angle given LST and Right Ascension
  $HA = $LST - $CRA; #Compute hour angle in degrees
  $HA = $HA + 180.0 while ($HA < -180.0);  #Make sure HA >= -180
  $HA = $HA - 180.0 while ($HA > 180.0);   #Make sure HA <= 180
  $HA = $HA*$RAD;    #HA in radians

#4. Compute ALT-AZ coordinates
  $y = sin($DEC)*sin($LAT) + cos($DEC)*cos($LAT)*cos($HA);    #Compute altitude
  $y = 1.0 if (1.0 - $y**2 <= 1.0E-10);   #Round-off error
  $x = sqrt(1.0 - $y**2);
  $ALT = atan2($y, $x); #ARCSIN

  $AZ = sin($DEC) - sin($ALT)*sin($LAT); #Compute azimuth
  $AZ = $AZ/(cos($LAT)*cos($ALT));
  $AZ = $PI/2.0 - atan2($AZ, sqrt(1.0 - $AZ**2));  #ARCCOS
  $AZ = 2.0*$PI - $AZ if (sin($HA) > 0.0);

  $ALT = $ALT/$RAD;   #Convert altitude to degrees
  $AZ = $AZ - 2.0*$PI if ($AZ >= 2.0*$PI);
  $AZ = $AZ/$RAD;   #Convert azimuth to degrees
} ###############################################################################

sub check_polaris { ##########################################################
#This subroutine reads the output file from the Polaris monitor and determines if
#the sky transparency is good enough to do observations.
#

#Reads the file created by the polaris monitor program.  (Reads each line into an
#element of the array 'monitor.')
  $polaris_file = 'C:\SLIC\\DATA\\'.$date_name.'\\'.$date_name.'.POL';   #Name of file
  if (-s $polaris_file) { # See if file exists; is > 0 bytes
    open (POLARIS, $polaris_file); # open file if it exists
    $. = 0;
    do {$line = <POLARIS> } until $. == desired_line_number || eof;
    chomp $line;   #Cut off return character
    seek(POLARIS, 0, 1);  #Goto EOF
    close (POLARIS)}
  else {
    &tasks_done("UNABLE TO LOCATE POLARIS MONITOR FILE, $polaris_file.\n");
    &shut_down}

#The intensity of Polaris = the last entry on line
  $intensity = substr($line, 76, 8);
  close (POLARIS); # close file
  if ($intensity <= 2500.00){ #If intensity is below some limit.
    &tasks_done("* * * WEATHER CONDITIONS PROHIBIT OBSERVING. * * *\n");
    &shut_down}    #Power down and wait until the next evening
} ###############################################################################

sub check_RA_DEC { ###########################################################
   $tolerance = 5;
   print "\nChecking if coordinates are correct...\n";
   if (  (($current_RAh == $RAh) && ($current_DECd == $DECd))  # RA hour & DEC degrees
must be exact
 && (($current_RAm == $RAm) && ($current_DECm == $DECm)) )  # RA & DEC minutes must
be exact
```

```
   { #begin if...
$slew_error = 0;   # reset error counter
print "\nCoordinates are correct.\n";
print "\n\n";
print "\nRA  $current_RAh:$current_RAm:$current_RAs DEC
$current_DECd:$current_DECm:$current_DECs\n";
# &tasks_done ("RA $current_RAh:$current_RAm:$current_RAs   DEC
$current_DECd:$current_DECm:$current_DECs");
sleep(2);
return;
    } # end if...
   else  {  print "\n*** MOUNT MALFUNCTION ***\n";
 print "COULD NOT SLEW TO $RAh:$RAm:$RAs RA, $DECd:$DECm:$DECs DEC.\n";
 print "Going to next field.\n";
 $pic_num = $num_pics;   # Go to next field
 $mount_error = 'error'; # ERROR did not reach coordinates: take action!
 print "\nslew_error = $slew_error";
    } # end else
} ###############################################################################
# end sub check_RA_DEC

sub check_slew_reply { ######################################################
  # ----- Send and check command to slew to selected field ----- #

   $slew_error2 = 0;
   # This block of 'if' statements is to check the response of the Gemini mount to
   # the slew command ':MS#'.
   if ($response == 0 )   # ******** Response '0'  if command was accepted
********
 { $mount_error = 'none';
$error = 0;
&slew_monitor;     # Watches RA and DEC until two consecutive values are equal
&check_RA_DEC;     # Checks that the current RA and DEC are the desired RA and DEC
return;
  }# end if ($response == 0 )
   elsif ($response == 1 )  # ******* Response is '1Object below horizon.#'
*********
 { $response_count = 22;  # Get remaining response from gemini.
($count_in, $response) = $mount_port->read($response_count);
$mount_error = 'error';   # ERROR slewing mount: take action!
&tasks_done ("whoops...$response\n Try another field.");  # Print response
$pic_num = $num_pics;  # Goes to next field
return;
  }# elsif ($response == 1 )
   elsif ($response == 2 )  # Gemini sends two strings that start with 2...
 { $response_count = 1;   # Get next character to determine response.
($count_in, $response) = $mount_port->read($response_count);
if ($response eq "T")  # ******** Response is '2Telescope is not
aligned.#' *******
{ $response_count =  23;  # Get remaining response from gemini.
  ($count_in, $response) = $mount_port->read($response_count);
  $mount_error = 'error';  # ERROR slewing mount: take action!
  &tasks_done("*** MOUNT MALFUNCTION ***");
  &tasks_done("Ooops...T$response");
  &shut_down;
 }# end  if ($response eq "T")
elsif ($response eq "N")  # ******** Response is '2No object selected.#' ********
{ $response_count = 19;  # Get remaining response from gemini.
($count_in, $response) = $mount_port->read($response_count);
$error++;
if ( $error <= 3 )   # Try to slew 3 times if '2No object
selected.#' occurs.
{ &tasks_done("\nOoops...N$response.  Trying again...");
  $mount_error = 'error';  # ERROR receiving coordinates:
```

```
take action!
  return;
 }# end if ( ++$error <= 3 )
else
{ $mount_error = 'error';  # ERROR slewing mount: take action!
  &tasks_done("\n*** MOUNT MALFUNCTION ***");
  &tasks_done("\n*** Uh-oh... N$response ***\n");
  &shut_down;
 }# end else
 }# end elsif ($response eq "N")
else
{ $mount_error = 'error';
&tasks_done("*** MOUNT MALFUNCTION ***");
&tasks_done("Slew command not recognized.");
&shut_down;
 } # end else
  } # end elsif ($response == 2 )
   elsif ($response == 3 )  # ******* Response is '3Manual control.#'
*********
 {  $response_count = 16;  # Get remaining response from gemini.
 ($count_in, $response) = $mount_port->read($response_count);
 $mount_error = 'error';  # ERROR slewing mount: take action!
 &tasks_done("*** MOUNT MALFUNCTION ***");
 &tasks_done("Error slewing mount. $response.");
 &shut_down;
  } # end elsif ($response == 3 )
   else
 {  $mount_error = 'error';
 &tasks_done("*** MOUNT MALFUNCTION ***");
 &tasks_done("Slew command not recognized.");
 &shut_down;
  } # end else
  &tasks_done("Slew complete.");  #Write to file when complete.
} ###########################################################################
# end sub send_and_check_command

sub configure_ports { ########################################################
#This subroutine contains the baudrate, port and various other information needed
#for PERL to communicate with the devices attached to the serial ports.
# They are the stamps used to control the filter, mechanical valve for
# the dewar refill and camera focus;  the stamp used to control the roof; and
# the Gemini telescope mount.
# last modified 11/6/02
# Configure Serial Port used for mount control.
# Serial Port #1(COM1), name is mount_port.
 $mount_port = new Win32::SerialPort ("COM1") || die "Can't Open $PortName: $^E\n";
    # serial ports name is $mount_port
 $mount_port->baudrate(9600)    || die "bad baudrate";
 $mount_port->parity('none')    || die "bad parity";
 $mount_port->databits(8)    || die "bad databits";
 $mount_port->stopbits(1)    || die "bad stopbits";
    #  $mount_port->buffers(1024,1024);
    #  Default timeouts
 $mount_port->read_char_time(0);
 $mount_port->read_const_time(1000);   # read command waits 5 seconds
 $mount_port->read_interval(0);
 $mount_port->write_char_time(0);
 $mount_port->write_const_time(3000);
 $mount_port->handshake("none");
 $mount_port->write_settings    || undef $mount_port;
 unless ($mount_port) {die "couldn't write_settings";}

# ***********
# Configure Serial Port used to communicate with the BSIIe controlling the
```

```
# focuser, filter & dewar valve. Serial Port #2(COM2), name is ffd_port  -->
# Focus/Filter/Dewar port.
  $ffd_port = new Win32::SerialPort ("COM2") || die "Can't Open $PortName: $^E\n";
  $ffd_port->baudrate(2400)   || die "bad baudrate";
  $ffd_port->parity('none')   || die "bad parity";
  $ffd_port->databits(8)   || die "bad databits";
  $ffd_port->stopbits(1)   || die "bad stopbits";
       # $ffd_port->buffers(1024,1024);
       # Default timeouts
  $ffd_port->read_char_time(0);
  $ffd_port->read_const_time(600000); # read command waits 10 minutes, 600000 ms
  $ffd_port->read_interval(0);
  $ffd_port->write_char_time(0);
  $ffd_port->write_const_time(3000);
  $ffd_port->write_settings   || undef $ffd_port;
  unless ($ffd_port) {die "couldn't write_settings"}

# ***********
# Configure Serial Port used to communicate with the BSIIe that will control the roof.
# Serial Port #5(COM5), name is roof_port.
$roof_port = new Win32::SerialPort ("COM5") || die "Can't Open $PortName: $^E\n";
$roof_read_time = 600000;
&configure_roof_port;
} ##############################################################################
# end sub configure_ports

sub configure_roof_port { ##########################################################
# Configure Serial Port used to communicate with the BSIIe that will control the roof.
# Serial Port #8(COM8), name is roof_port.
# This routine is seperate so the read time for the roof port
# can be changed mid-program.  The read time can be changed by first
# setting the variable $roof_read_time to the desired amount
# of milliseconds, then call this subroutine.
# last modified 11/06/02    K.P. Portock
  $roof_port->baudrate(2400)    || die "bad baudrate";
  $roof_port->parity('none')    || die "bad parity";
  $roof_port->databits(8)    || die "bad databits";
  $roof_port->stopbits(1)    || die "bad stopbits";
#  $roof_port->buffers(1024,1024);
#  Default timeouts
  $roof_port->read_char_time(0);
  $roof_port->read_const_time($roof_read_time); # read command waits a variable amount of
seconds
  $roof_port->read_interval(0);
  $roof_port->write_char_time(0);
  $roof_port->write_const_time(3000);
  $roof_port->write_settings    || undef $roof_port;
  unless ($roof_port) {die "couldn't write_settings"}
#  sleep(5);  #Maybe configuring the ports needs some time...
} ##############################################################################
# end sub configure_roof_port#

sub cool_down { ####################################################################
#Wait to give camera sufficient time for cool-down.
#
  $roof_status_variable = 0; #Print some roof status messages only once
  &tasks_done("Camera cool-down in progress... Cooling ends in 2.0 hours.");
  print "\n";  #Print blank line to screen
  @now = gmtime;   #Current time
  $cool = 1/30 + DECIMAL(1, $now[2], $now[1], $now[0]); #<------------ 2.0 + DECIMAL(1, $now[2],
$now[1], $now[0]);
  if ($cool >= 24.0) {  #If cooling period goes past midnight
    $cool = time_range($cool);  #Make hour 0 - 24
    $now[3] = $now[3] + 1.0;  #We are in a new day
```

```
  }
  @cool = decimal_converter($cool); #End of cooling period, in HH:MM:SS
  $JD_cool = Julian_day($now[3], $now[4]+1.0, $now[5]+1900.0, $cool, 0, 0);
#Julian day for 2 hours after end of dewar fill.
  $print_status = $JD_cool - 1/12;

  do {
    @now = gmtime; $JD_now =
  Julian_day($now[3], $now[4]+1, $now[5]+1900, $now[2], $now[1], $now[0]);
    $status = $JD_now - $print_status;
    if ($status > 1.0/96.0 || $status == 0) {  #Print status every 15 minutes
  printf "\nCurrent time is %02d:%02d:%02d UT. ", $now[2], $now[1], $now[0];
  printf "Cooling period ends @ %02d:%02d:%02d UT.", $cool[0], $cool[1], $cool[2];
  $print_status = $JD_now;  sleep(1.5);     # Reset counter; wait 1.55s
    }
    sleep(5);
    &roof(3);   # Ping roof to get status. Roof will timeout after 20 minutes
  } while ($JD_now < $JD_cool);
  &tasks_done("Camera cool-down complete...\n");
} ##############################################################################

sub copy_files { ###################################################################
#Uses the DOS 'copy' command to copy the evening's directory from SLIC to CD and
#ASTRO.
#
  $new_directory = "C:\\PHIL\\".$date_name; #Name of the back-up directory<------- CHANGE IN
FINAL VERSION
  mkdir($new_directory);    #Make the new directory
  $astro_dir = "slic@astro.phys.vt.edu:data"; # Astro directory to store files

  &tasks_done("Copying OBS file, $input, to $directory.");
  copy ($input, $directory); #Copy OBS file to the observing directory
  &tasks_done("$input copy successful.\n");

  $obs_dir_size = 0;  $dir_size = 0;   #Initialize counters
  opendir(OBS_DIR, $directory);    #Open observing directory
  while (defined($file_name = readdir(OBS_DIR))) {  #Get names of all files in obs directory
next if $file_name =~ /^\.\.?$/;
$file_name = $directory."\\".$file_name;
@file_stats = stat($file_name);      #Get file statistics; mainly size
$obs_dir_size = $obs_dir_size + $file_stats[7];  #Add file sizes
  }
  closedir(SUB_DIR);
  @file_stats = stat("C:\\SLIC\\".$date_name."_OBS.txt"); #How large is OBS file?
  $obs_dir_size = $obs_dir_size + $file_stats[7];
  @file_stats = stat("C:\\SLIC\\".$date_name.".POL");     #How large is POL file?
  $obs_dir_size = $obs_dir_size + $file_stats[7];     #Total directory size

  opendir (DIR, $new_directory);
  while (defined($new_dir = readdir(DIR))) {  #Open back-up directory
$sub_directory = "A:\\".$new_dir;
opendir(SUB_DIR, $sub_directory);     #Open all sub-directories in
it
while (defined($file_name = readdir(SUB_DIR))) {     #Get names of all sub-dir.
next if $file_name =~ /^\.\.?$/;
$file_name = $sub_directory."\\".$file_name;
@file_stats = stat($file_name);
$dir_size = $dir_size + $file_stats[7];   #Get size of all files in back-up directory
}
closedir(SUB_DIR);
  }
  closedir(DIR);
  $dir_size = 555155456 - $dir_size;  #space remaining in back-up directory
  if ($dir_size > $obs_dir_size) {
```

```
    $log_file = "C:\\SLIC\DATA\\".$date_name."\\".$date_name.".LOG";
    &tasks_done("Copying observing directory $directory to $new_directory.");
    opendir (DIR, $directory);  #Open the observing directory
    while (defined($file_name = readdir(DIR))) { #read file names
 next if $file_name =~ /^\.\.?$/;    # skip . and ..
 $file_name = "C:\\SLIC\DATA\\".$date_name."\\".$file_name;
 copy($file_name, $new_directory); #Copy obs. directory to the back-up
 &tasks_done("$file_name copied to $new_directory.");
    }
    closedir (DIR);
    &tasks_done("$directory successfully copied to $new_directory.\n");
    copy($log_file, $new_directory);     #Recopy updated log file to back-up

    &tasks_done("Copying observing directory $directory to $astro_dir.");
    system "scp2 $directory $astro_dir";      #Copy obs. directory to Astro
    &tasks_done("$file_name copied to $astro_dir.");
  }
  else {
    &tasks_done("**** WARNING!!! BACK-UP DIRECTORY IS FULL. ****");
    $status_msg = "(Insert a new disk and MANUALLY copy $directory, $input ";
    $status_msg = $status_msg."and $polaris_file to $new_directory.)\n"
    &tasks_done($status_msg);
  }
} ###########################################################################

sub create { ########################################################################
#Creates a unique label - using the format YY_MM_DD.  It then uses this label to
#create and name the directory where the evening's observations will be stored. It
#also creates the log file - YY_MM_DD.LOG - for the evening's observing session.
#
#Create the name label
  @date = localtime; #Reads local time from system clock
  $year = $date[5] - 100.0; #2-digit year
  $year4 = $year + 2000.0; #Some calculations require a 4-digit year
  $month = $date[4] + 1.0; #month
  $day = $date[3]; #day of month
  substr($month,0,0) = '0' x (2-length($month)); #Gives 2-digit month
  substr($day,0,0) = '0' x (2-length($day));  #Gives 2-digit day
  substr($year,0,0) = '0' x (2-length($year));  #Gives 2-digit year
  $date_name = $year.'_'.$month.'_'.$day;  #Directory/log file name

#Create the directory
  $directory = 'C:\SLIC\DATA\\'.$date_name;     #name of directory: C:\SLIC\YY_MM_DD
  mkdir($directory);  #create evening's directory if it doesn't exist
  chdir($directory);  #makes this the current directory

#Create log file named YY_MM_DD.LOG
  $log_file = $date_name.'.LOG';
  open(LOGFILE,">>$log_file") || die("Cannot Open File!");
  close (LOGFILE);  #Close log_file
} ###########################################################################

sub DECIMAL { ########################################################################
#Convert XX:MM:SS to decimal XX. X = hours or degrees; M = (arc)minutes;
#S = (arc)seconds.
#
  my($sign, $XX, $MM, $SS) = @_;
  $XX = abs($XX);  $MM = abs($MM);  $SS = abs($SS); #make all numbers +
  $value = $XX + ($MM + $SS/60.0)/60.0;     #decimal value
  return $value*$sign;    #supply any '-' signs
} ###########################################################################

sub decimal_converter { ##############################################################
#Convert from decimals back to DD/HH:MM:SS.
```

```
#
  my($DATA, @UNITS) = @_;
  if ($DATA < 0) {
    $sn = -1;
    $DATA = abs($DATA)}
  else {$sn = 1}
  $UNITS[0] = int($DATA);    #Gives degrees/hours

  $temp1 = $DATA-$UNITS[0];
  $temp2 = 60.0*$temp1;
  $UNITS[1] = int($temp2);   #Gives minutes

  $temp3 = $temp2-$UNITS[1];
  $UNITS[2] = 60.0*$temp3;   #Gives seconds

  if ($UNITS[2] > 59) {     #Make seconds < 60
    $UNITS[2] = 0; $UNITS[1] = $UNITS[1] + 1}
  if ($UNITS[1] > 59) {     #Make seconds < 60
    $UNITS[1] = 0; $UNITS[0] = $UNITS[0] + 1}

  if ($UNITS[0] ne 0) { #This 'if' block puts negative signs
    $UNITS[0] = $sn*$UNITS[0]} #in front of the first non-0 element
  else {   #if a number is < 0
    $UNITS[1] = $sn*$UNITS[1] if ($UNITS[1] ne 0);
    $UNITS[2] = $sn*$UNITS[2] if ($UNITS[2] ne 0 && $UNITS[1] eq 0)}
  return @UNITS;
} ###############################################################################

sub dewar { ###################################################################
#PERL dewar control program. Use with focus_filter_dewar3.bs2. Fills camera dewar.
#
  @coordinates = hor2eq(3, 270);    # Point camera west for filling
  &slew(@coordinates);
  &mount(park);

#Compute Julian day number for when dewar fills
  @dewar_fill = gmtime;   #Universal time that dewar fill begins
  $dewar_fill = Julian_day($dewar_fill[3], $dewar_fill[4] + 1, #Julian date
   $dewar_fill[5] + 1900, $dewar_fill[2], $dewar_fill[1], $dewar_fill[0]);
  $JD_status = $dewar_fill;  #Counter used to print status message

  open (LOGFILE, ">>$log_file");     #Append log
  print (LOGFILE "\n");      #Print blank line to log
  close(LOGFILE);
  &tasks_done("Dewar refill begun.");    #Print this comment to log file

  $roof_status_variable = 0; #Print some roof status messages only once
  $start_fill = 3;
#This is the value the BASIC stamp is waiting for - via SERIN cmd -
#to start filling the camera dewar.
  $ffd_port->write($start_fill);     #Tells the BASIC stamp to refill dewar.
  ($dummy_variable, $pin_test) = $ffd_port->read(1); #Read Stamp echo

  do {
#Compute Julian day number for current time
@too_long = gmtime;
$too_long = Julian_day($too_long[3], $too_long[4] + 1.0,
  $too_long[5] + 1900.0, $too_long[2], $too_long[1], $too_long[0]);

($dummy_variable, $pin_10) = $ffd_port->read(1); #Read the state of pin 10.

if ($too_long > $dewar_fill + 1.0/24.0) {
$ffd_port->write(2);  #This forces the Stamp to close the valve
($dumb, $dumber) = $ffd_port->read(1); #read stamp echo
```

```
&tasks_done("Dewar refill has exceeded 30 minutes.  Shutting down...\n");
&shut_down}  #Shut down if refill goes beyond 1/2 hour (1/48 of a day)

    @now = gmtime; $JD_now =    #Current time
  Julian_day($now[3], $now[4]+1, $now[5]+1900, $now[2], $now[1], $now[0]);
    $status = $JD_now - $JD_status;
    if ($status > 1.0/960.0) {   #Print status message every 1.5 minutes
  print "\n", scalar(gmtime), " UT- Dewar refill in progress...";
  $JD_status = $JD_now; sleep(5);
    }
    sleep(2);  # <-------  MAY NEED TO MODIFY OR REMOVE
    &roof(3);   # Ping roof to get status. Roof will timeout after 20 minutes
  } while ($pin_10 == 0);

  sleep(5); @end_fill = gmtime;   # Time that dewar fill ends
  $old_fill = &DECIMAL(1.0, $end_fill[2], $end_fill[1], $end_fill[0]);
  $old_fill = Julian_day($end_fill[3], $end_fill[4]+1.0, $end_fill[5]+1900.0, $old_fill, 0, 0);
#Convert the time of this dewar fill to hours
  &tasks_done("Dewar refill complete...\n\n");  # Print this comment to log file
  &mount(track);
  manual_slew(neutral);   # Point back to neutral position
} ##############################################################################

sub filter { ###################################################################
# This subroutine communicates with the BASIC stamp to change
# the camera filters. Use with focus_filter_dewar3.bs2
# Modified 7/8/02 K.P. Portock
#
  my($filter_goal) = @_;
  $missed = 0;

  &roof(3); # Ping roof to get status. Roof will timeout after 20 minutes

  Filter_top:  # Label for goto statement
    $ffd_port->write(2);  # Tell stamp to goto filter routine by sending a "2".
    ($count_in, $echo) = $ffd_port->read(1);  # Echo

    $ffd_port->write($filter_goal);  # Tells stamp to go, sends desired filter
number
    ($count_in, $echo) = $ffd_port->read(1);  # Echo
    &tasks_done("Turning filter wheel...");  # Write to log file

    ($count_in, $filter) = $ffd_port->read(1);# Stamp tells PERL which filter is engaged
    &tasks_done("filter = $filter");  # Print to file
    if ($filter == 0)  # If PERL gets a zero from the stamp then the
filter wheel has malfunctioned
  { &tasks_done("**** FILTER WHEEL MALFUNCTION! ****");
 &shut_down;
} # end if ($filter == 0)
    elsif ($filter == $filter_goal)  # Check if engaged filter is the correct filter
{  $missed = 0;   }  # Filter is in correct position  reset error counter
    else
{  # if filter is not in correct position, try again. Try a max of 3 times.
   &tasks_done("Did not reach the desired filter.");
   $missed++;  # Count number of attempts to get to the
correct filter.
   if ($missed >= 3)  # If tried 3 times to get to filter and never
made it then shut down.
 { &tasks_done("*** FILTER WHEEL MALFUNCTION! ***");
&shut_down;  # Stamp tries 5 times it detects an error
  } # end if ($missed >= 3)
   &tasks_done("Trying again to reach filter...\n");
   goto Filter_top;  # Try again by going to Top
 } # end else
```

```
   # Each filter has a different focus
    if ($filter == 1) { &focus("0808") }  # Goto focus #1
    if ($filter == 2) { &focus("4253") }  # Goto focus #2
<-------------these values for testing
    if ($filter == 3) { &focus("2560") }  # Goto focus #3
purposes only.
    if ($filter == 4) { &focus("8701") }  # Goto focus #4
    return;   # continue observations

} ##############################################################################

sub focus { ####################################################################
# Sends intructions to BASIC STAMP II to turn stepper motor.
# Stepper motor attached to camera lens by a worm/gear set up to rotate lens
# to the desired focus.
# 4-digit instruction:
# 4 digits select new position to turn to.
# Set new position by entering 4 digits in qoutes.
# Ex: For position #200, $new_pos = "0200".  For position 1, $new_pos = "0001".
# To go back to original starting position set $new_pos = "0000"
#   (position number 0) Stamp does NOT look for "home" position.
#   *** NOTE *** When Stamp program starts after being shut down it assumes the motor is in
# position 0000.
#
# One full rotation of stepper motor = 200 steps -> 1 degree of lens rotation
# If a larger range of operation is desired can set new position to 5 digits; must also
# edit focus routine in focus_filter_dewar3.bs2 to read 5 digits.
#
# Use with Stamp program: focus_filter_dewar3.bs2
# Modified 7/8/02 K.P. Portock, Added focus file routines (12/10/02) - KPN

  my($new_pos) = @_;  # New position input when focus routine is called
  $focus_error = "none";  # Reset error flag

  if ($roof_error ne "error") {&roof(3)} # Ping roof to get status. Roof will timeout
after 20 minutes
  $ffd_port->write(1);  # Tell stamp to goto focus routine by sending a
'1'.
  ($count_in, $echo) = $ffd_port->read(1);  # echo

  $ffd_port->write($new_pos);  # Tell stamp ew position to turn to.
  ($count_in, $echo) = $ffd_port->read(4);  # echo
  &tasks_done("Focuser turning to postion = $echo");

  ($count_in, $pos_in) = $ffd_port->read(4);  # Wait to read new postion from STAMP
 # 4 digits reply from Stamp
  if (length($pos_in) < 4 )  # If a 4 digit reply is not read then there is
an error
   { &tasks_done("*** FOCUSER MALFUNCTION! ****");  # write to file
$focus_error = "error";   # Raise error flag
&shut_down;  # shut down
    } # end if (length($pos_in) < 4 )

  &tasks_done("New Postion $pos_in confirmed."); # Write to file
  open (FOCUS, ">$focus_setting");  # open focus.txt; create if doesn't exist
  print (FOCUS $new_pos);   # Print new focusser position to file
  close(FOCUS);
} ##############################################################################

sub format_coord { #############################################################
 # formats RA and DEC coord
 #
   my(@position) = @_;
```

```
 #formats coordinates as two digits - i.e., HH:MM:SS or DD:MM:SS so they can be read
 #by the gemini mount. SS are rounded UP (55.8 -> 56).
   substr($position[0],0,0) = '0'x(2-length($position[0])); $RAh = $position[0];
   substr($position[1],0,0) = '0'x(2-length($position[1])); $RAm = $position[1];
   $RAs = int($position[2] + 0.5); substr($RAs,0,0) = '0'x(2-length($RAs));

  if (($position[3] < 0) || ($position[3] == 0 && $position[2] < 0) ||
 ($position[3] == 0 && $position[2] == 0 && $position[2] < 0)) {
$sign = -1}
  else {$sign = 1}
  if ($sign < 0) {
if ($position[3] == 0) {$position[3] = '-00'}
else {
  $position[3] = abs($position[3]);
  substr($position[3],0,0) = '0'x(2-length($position[3]));
  $position[3] = '-'.$position[3];
  }
   }
   else {substr($position[3],0,0) = '0'x(2-length($position[3]))}
   $DECd = $position[3];
   $position[4] = abs($position[4]) if ($sign < 0 && $position[3] == 0);
   $position[5] = abs($position[5]) if ($sign < 0 && $position[3] == 0 && $position[4] == 0);
   substr($position[4],0,0) = '0'x(2-length($position[4])); $DECm = $position[4];
   $position[5] = int($position[5] + 0.5);
   substr($position[5],0,0) = '0'x(2-length($position[5])); $DECs = $position[5];
} ##############################################################################
# end sub format_coord

sub HA_15min { ###################################################################
#For those fields we don't want to observe in the Eastm wait until they cross
#-15 minutes of hour angle

  $count_ha = 0;    # counter
  @now = gmtime; $print_status =    # Julian date for current time
     Julian_day($now[3], $now[4]+1, $now[5]+1900.0, $now[2], $now[1], $now[0]);

  do {        # This loop waits until HA >= -15 min.
    &altitude;        # Compute the altitude, HA of the field
    &time_to_quit;        # Has observing gone beyond end time?
    @now = gmtime; $now =        # Julian date for current time
  Julian_day($now[3], $now[4]+1, $now[5]+1900.0, $now[2], $now[1], $now[0]);
    if ($now - $print_status >= 1.0/288.0) {  # Prints message every 5 minutes
      @HA = &decimal_converter($HA/(15.0*$RAD));  #Hour Angle in HH:MM:SS
      $HA[0] = "0".$HA[0] if ($HA[0] < 10 && $HA[0] >= 0);
      $HA[1] = "0".$HA[1] if ($HA[1] < 10 && $HA[1] >= 0);
      $HA[2] = "0".$HA[2] if ($HA[2] < 10 && $HA[2] >= 0);
      if ($HA < 0) {$HA[0] = "-".$HA[0]}
      &tasks_done("$field is in the East (HA = $HA[0]:$HA[1]:$HA[2]).");
      &tasks_done("Waiting for it to cross meridian.\n");
      $print_status = $now;       # counter reset
    }
    ++$count_ha; # increment counter
  } while ($HA < -3.75*$RAD); # Wait until HA >= -15 min.
} ##############################################################################

sub header_info { ################################################################
#Print name of observer, start/stop times and maximum time that can be used to
#observe to log file
#
  open (LOGFILE, ">>$log_file");     #Append log
  print (LOGFILE "\t\t\t\tOBSERVING LOG - $date_name\n");
  print (LOGFILE "\t\t\t\t=======================\n\n");
  print (LOGFILE "OBSERVER:  $inputdata[0]\n"); #Prints observer's name to log
```

```
  $start_hour = time_range($starting[0]);    #Actual start time is 0 - 24
  print (LOGFILE "\nActual observing begins approximately at ");
  printf (LOGFILE "$start_tag which occurs at %02dh %02dm UT.",
 $start_hour, $starting[1]);
  print (LOGFILE "\nObserving will end by $end_tag ");
  printf (LOGFILE "which occurs at %02dh %02dm UT.",
 $ending[0], $ending[1]);
  printf (LOGFILE "\nMAXIMUM OBSERVING TIME: ");
  printf (LOGFILE "%02dh %02dm.\n", $duration[0], $duration[1]);
  close (LOGFILE);
} ##############################################################################

sub hor2eq { ###################################################################
#Determines the RA and DEC to point the telescope for filling. Computations were
#adapted from the book, _Practical Astronomy With Your Calculator, 3rd Edition_,
#by Peter Duffett-Smith, pg. 38. (Used to determine the RA and DEC to point the
#telescope for filling and for taking correction images.)
#
  my($altitude, $azimuth) = @_;

#Convert ALT/AZ from degrees to radians.
  $altitude = $altitude * $RAD;  $azimuth = $azimuth * $RAD;

#Compute declination
  $DEC = sin($altitude)*sin($LAT) + cos($altitude)*cos($LAT)*cos($azimuth);
  $DEC = atan2($DEC, sqrt(1.0 - $DEC**2));
  @DEC = decimal_converter($DEC/$RAD);

  $x = sin($altitude) - sin($LAT)*sin($DEC);  #Hour Angle
  $y = -sin($azimuth)*cos($LAT)*cos($altitude);
#  $y = 0.0 if (abs($y) < 1.0E-15);
  $H_A = atan2($y, $x);   #HA in radians
  $H_A = $H_A/(15.0*$RAD);  #HA in hours

#Compute Right Ascension
  @time = gmtime;    # current UT
  $UT = DECIMAL(1, $time[2], $time[1], $time[0]); #Decimal UT
  $UT_day = $time[3]; $UT_mon = $time[4]+1; $UT_yr4 = $time[5]+1900;
  $LST = UT2LST($UT, $UT_day, $UT_mon, $UT_yr4);   #Get LST from UT
  $RA = $LST - $H_A;  #Right ascension
  $RA = time_range($RA);    #RA should be bewt. 0 & 24
  @RA = decimal_converter($RA);

  return (@RA, @DEC);
} ##############################################################################

sub initialize_focus { #########################################################
# Initializes focus stepper-motor
  $focus_setting = 'C:\\SLIC\\DATA\\focus.txt';

  if (-s $focus_setting) {       # File exists and has a size > 0?
    open(FOCUS, "<$focus_setting");  # Open file if it exists
    $old_focus = <FOCUS>;       # Reads previous focus value from file
    close (FOCUS);       # Closes file
    substr($old_focus,0,0) = '0' x (4-length($old_focus)); #Gives 4-digit focus value

   $ffd_port->write(4);  # Tell stamp to goto init focus routine by
sending a '4'.
  ($count_in, $echo) = $ffd_port->read(1);  # echo

  $ffd_port->write($old_focus);  # Tell stamp ew position to turn to.
  ($count_in, $echo) = $ffd_port->read(4);  # echo
  &tasks_done("Focuser set to position = $echo");
  }
```

```
  else {$old_focus = "0000"}   # Otherwise, focus is at 0000
} ###########################################################################

sub initialize_mount { ###################################################
# initializes gemini mount setting.  Time, time zone, lattitude & longitude.

    #-------Get geminis date and time--------#
    $mount_string = 'date';
    $response_count = 9; # Number of characters in Geminis reply.  Reply depends
on command sent.
    $mount_command = ":GC#"; # Command to GET CALENDAR DATE from Gemini Mount
    &send_to_mount; # Send command
    $gem_mn = substr($response,0,2);   # Seperate gemini date into month day and year
    $gem_dd = substr($response,3,2);
    $gem_yr = substr($response,6,2) + 2000;

    $mount_string = 'Time';
    $response_count = 9; # Number of characters in Geminis reply.  Reply depends
on command sent.
    $mount_command = ":GL#"; # Command to GET TIME from Gemini Mount Displays
in GMT
    &send_to_mount; # Send command
    $gem_hh = substr($response,0,2)+ 5;  # Seperate Gemini time into hours minutes and seconds
    $gem_mm = substr($response,3,2);
    $gem_ss = substr($response,6,2);
    $gem_UT = DECIMAL(1,$gem_hh,$gem_mm,$gem_ss);  # Actual UT from PC
    $gem_LST = UT2LST($gem_UT, $gem_dd, $gem_mn, $gem_yr);    # Actual LST calculated from
actual UT

    #--- Get date and time from PC.  Format date and time. ----#
    @date = gmtime;  # Reads UT & date from system clock
    $PC_yr = $date[5] + 1900;  # 4-digit year
    $PC_mn = $date[4] + 1;  # Month
    $PC_dd = $date[3];   # Day of month
#    substr($month,0,0) = '0' x (2-length($month));   # Gives 2-digit month
#    substr($day,0,0) = '0' x (2-length($day));  # Gives 2-digit day
#    substr($year,0,0) = '0' x (2-length($year));    # Gives 2-digit year
#    $date = "$month/$day/$year"; # Date in mm/dd/yy format

#    $local_time = localtime;
#    $gm_time = gmtime; @gm_time = gmtime;
#    $time_zone = 5;  # Calculates difference in local time and GMT in hours to get time zone
    $PC_hh = $date[2];  # Get hour
    $PC_mm = $date[1];        # Get minute
    $PC_ss = $date[0];        # Get second
#    substr($hour,0,0) = '0' x (2-length($hour));    # Gives 2-digit hour
#    substr($minute,0,0) = '0' x (2-length($minute)); # Gives 2-digit minute
#    substr($second,0,0) = '0' x (2-length($second)); # Gives 2-digit second
    $Real_UT = DECIMAL(1,$PC_hh,$PC_mm,$PC_ss);  # Actual UT from PC
    $Real_LST = UT2LST($Real_UT, $PC_dd, $PC_mn, $PC_yr);   # Actual LST calculated from actual
UT

    $delta_HA = $Real_LST - $gem_LST;   # Hour angle difference - decimal
    @delta_HA = decimal_converter($delta_HA);   # Hour angle difference - hh:mm:ss
    substr($delta_HA[0],0,0) = '0' x (2-length($delta_HA[0]));  # Gives 2-digit hour
    substr($delta_HA[1],0,0) = '0' x (2-length($delta_HA[1]));  # Gives 2-digit min
    $delta_HA[2] = int($delta_HA[2] + 0.5);  # Round to whole seconds
    substr($delta_HA[2],0,0) = '0' x (2-length($delta_HA[2]));  # Gives 2-digit sec
    &tasks_done("Difference in Gemini time and real time:
$delta_HA[0]:$delta_HA[1]:$delta_HA[2]");
    &tasks_done("delta_HA = Real_LST - gem_LST");

    &mount(track);
```

```
    #$mount_string = 'Time zone set';
    #$response_count = 1; # Number of characters in Geminis reply.  Reply depends
on command sent.
    #$mount_command = ":SG$time_zone#";  # Form proper LX200 command to SET TIME ZONE for
Gemini Mount
    #&send_to_mount; # Send command

    #$mount_string = 'Local time set';
    #$response_count = 1; # Number of characters in Geminis reply.  Reply depends
on command sent.
    #$mount_command = ":SL$l_time#"; # Form proper LX200 command to SET LOCAL TIME
for Gemini Mount
    #&send_to_mount; # Send command

    #$mount_string = 'Calendar date set';
    #$response_count = 50; # Number of characters in Geminis reply.  Reply
depends on command sent.
    #$mount_command = ":SC$date#"; # Command to SET DATE for Gemini Mount
    #&send_to_mount; # Send command
#----Set observing site longitude and latitude-------#
    #$mount_string = 'Observing site longitude set';
    #$response_count = 1; # Number of characters in Geminis reply
    #$mount_command = ":Sg+080*33#"; # Command to SET LONGITUDE of observing site.
Format: +/-ddd*mm
    #&send_to_mount; # Long. of SLIC (80d 33m 30s W)

    #$mount_string = 'Observing site latitude set';
    #$response_count = 1; # Number of characters in Geminis reply
    #$mount_command = ":St+37*20#"; # Command to SET LATITUDE of observing site. Format:
+/-dd*mm
    #&send_to_mount; # Lat. of SLIC (37d 19m 48s N)
} ###########################################################################
# end sub initialize_mount

sub input { ###############################################################
#This routine reads the input file, YY_MM_DD_OBS.txt, and stores it in an array,
#inputdata, which the program uses to determine what fields to observe.  If this
#file does not exist, a message is printed to the logfile to indicate that this
#file could not be located.  SLIC.pl then waits until the new observing day.
#
  $input = 'C:\SLIC\\'.$date_name.'_OBS.txt';  #Name/location of observing file
  if (-s $input) {  #File exists and has a size > 0?
    open(INPUT, $input); #open file if it exists
    @inputdata = <INPUT>; #Creates an array of observing data
    close (INPUT)}  #Closes input file
  else {
    &tasks_done("UNABLE TO LOCATE *.OBS FILE, $input.\n"); #Print the error msg.
    &shut_down} #shut_down system and wait until new day
} ###########################################################################

sub Julian_day { ##########################################################
#Compute the Julian Day Number, including fraction of a day
#
  my($JD_day, $JD_mnth, $JD_yr, $JD_hr, $JD_min, $JD_sec) = @_;
  if ($JD_mnth < 3) { #If month is before March
    $JD_yr = $JD_yr - 1.0;
    $JD_mnth = $JD_mnth + 12.0}

  $A = int($JD_yr/100.0);
  $B = 2.0 - $A + int($A/4.0);
  $C = int(365.25*$JD_yr);
  $D = int(30.6001*($JD_mnth+1.0));
  $fraction = DECIMAL(1, $JD_hr, $JD_min, $JD_sec)/24.0;   #time - fraction of a day
```

```
  return $B + $C + $D + $JD_day + $fraction + 1720994.5;   #Julian date
} ###############################################################################

sub last_day { ##################################################################
#A subroutine used by the subroutine startime...
#
#Are we reading from the last day of the month?  For certain times (ex.:
#am, moonrise or moonset), we need to read the next day's data. If this
#occurs at the end of the month, the program needs to know to go to the
#first day of the next month.
#
  my($m, $y, $r, $clmn) = @_;
  @days_31 = (1, 3, 5, 7, 8, 10, 12);  #Month number of months with 31 days
  @days_30 = (4, 6, 9, 11);  #Month number of months with 30 days
  for ($entry = 0; $entry <= 6; ++$entry) {
    if (($m == $days_31[$entry] && $r == 44) ||
  ($m == $days_30[$entry] && $r == 43)) {$r = 13; $c = $clmn + 11}
    if ($m == 2) {    #February
  if ($y % 4 == 0 && $r == 42) {$r = 13; $c = $clmn + 11} #Leap year
  if ($y / 4 ne 0 && $r == 41) {$r = 13; $c = $clmn + 11} #Non-Leap year
    }
  }
  return ($r+1, $c);  #row and column
} ###############################################################################

#---------------------manual slewing procedure--------------------------#
sub manual_slew { ###############################################################
# This subroutine is used to 'manually slew' to desired coordinates that
# can not be slewed to in the usual 'automatic' way because the coordinates
# are below the horizon.  This sends MOVE NORTH, MOVE SOUTH, MOVE WEST, MOVE EAST,
# commands to the mount and monitors the slew until the coordinates are reached.
# can manually slew to four positions. 1 for each light box, and 1 for dewar refilling,
# and the fourth is the neutral position, counter weight down, just west of polaris.
# - If manual slew receives "dfill" it will slew camera to the dewar refill position
#   and park the mount.
#
  my($slew_type) = @_; # get desired slew type $slew_type = 'dfill' to put in position
to fill the dewar
#  $slew_type = 'westbox' to point to box
on west wall
#  $slew_type = 'eastbox' to point to box
on east wall
#  $slew_type = 'neutral' to point to
neutral position
  $response_count = 10;   # Number of characters in Geminis response.  Response
depends on command sent.
  $mount_string = 'move slew';  # For debuggin'
  $mount_command = ":RS#";  # Include this command before attempting to
slew manually
  &send_to_mount;

  print "\nPointing telescope to the neutral position\n";
  @coordinates = hor2eq(37.5, 357);  # Point camera north,
  &slew(@coordinates);
  sleep(1);  # Wait while motors stop

  if ($slew_type eq "dfill")  # Point to dewar filling position
{ print "\nSlewing to dewar refilling position.\n";
  @coordinates = hor2eq(-85, 270);  # Convert Alt/Az to RA/DEC
  # Do we want 0 270 for dewar filling position?
  # If we do then this whole subroutine could be eliminated
  # and monitor_manual_slew could be eliminated
  &format_coord(@coordinates);  # Format coordinates for monitoring routine
```

```
  $response_count = 0;
  $mount_string = 'Moving West';  # Move West
  $mount_command = ':Mw#';  # Form command to move telescope West(ra axis)
  &send_to_mount;  # Send command to mount
  &monitor_manual_slew(ra);  # Monitor the ra slew

  $response_count = 0;
  $mount_string = 'Moving North';  # Move North
  $mount_command = ':Mn#';  # Form command to move telescope north(dec
axis)
  &send_to_mount;  # Send command to mount
  &monitor_manual_slew(dec);  # Monitor the dec slew

  sleep(2);  # Wait while motors come to complete stop
  &mount(park);   # Stop telescope tracking

  return;  # Exit subroutine
} # end if ($slew_type eq "dfill")

  elsif ($slew_type eq "westbox")  # Point to light box on west wall
{ print "\nPointing telescope to the light box on the west wall.\n";
  @coordinates = hor2eq(5,270);   # Convert Alt/Az to RA/DEC
  &slew(@coordinates);
  sleep(1);  # Wait while motors stop
    # **** Below stuff commented out because above commands will work ****  #
 # $response_count = 0;
 # $mount_string = 'Moving West';   # Move West
 # $mount_command = ':Mw#';   # Form command to move telescope West(ra axis)
 # &send_to_mount;   # Send command to mount
 # &monitor_manual_slew(ra);   # Monitor the ra slew

 # $response_count = 0;
 # $mount_string = 'Moving North';   # Move North
 # $mount_command = ':Mn#';   # Form command to move telescope north(dec
axis)
 # &send_to_mount;   # Send command to mount
 # &monitor_manual_slew(dec);   # Monitor the dec slew

 # sleep(2);   # Wait while motors come to a complete stop
  &mount(park);    # Stop telescope tracking
  return;   # Exit subroutine
} # end if ($slew_type eq "westbox")
  elsif ($slew_type eq "eastbox")   # Point to lightbox on east wall
{ print "\nPointing telescope to the light box on the east wall.\n";
  @coordinates = hor2eq(0,90);   # Convert Alt/Az to RA/DEC
  &slew(@coordinates);
  sleep(1);   # Wait while motor stops
    # **** Below stuff commented out because above commands will work **** #

  # $response_count = 0;
  # $mount_string = 'Moving West'; # Move West
  # $mount_command = ':Mw#'; # Form command to move telescope West(ra axis)
  # &send_to_mount; # Send command to mount
  # &monitor_manual_slew(ra); # Monitor the ra slew

  # $response_count = 0;
  # $mount_string = 'Moving North'; # Move North
  # $mount_command = ':Ms#'; # Form command to move telescope north(dec axis)
  # &send_to_mount; # Send command to mount
  # &monitor_manual_slew(dec); # Monitor the dec slew

  # sleep(2); # Wait while motors come to a complete stop
  &mount(park);   # Stop telescope tracking
```

```
  return;  # Exit subroutine
} # end if ($slew_type eq "eastbox")
  elsif ($slew_type eq "neutral")
{ return; }  # Exit subroutine

} ##############################################################################
# end sub manual slew

#--------------------monitor manual slewing --------------------#
sub monitor_manual_slew { ##########################################################
# This routine monitors slewing when done manually.  Manually means giving
# the mount MOVE NORTH, MOVE SOUTH, MOVE EAST, MOVE WEST commands instead
# of giving it specific coordinates to slew to.
  my($direction) = @_;  # Gets direction to monitor from
command

  if ($direction eq "dec")  # Monitor the declination axis
{ for ($dec_check = 0; $dec_check <= 1; ++$dec_check)# Loop until the correct DEC is
reached
   { $dec_check = 0;
$response_count = 10;   # Number of characters in Geminis
response.  Response depends on command sent.
$mount_string = 'Current_DEC';   # For debuggin'
$mount_command = ":GD#";   # Form proper command to get the
current DEC
&send_to_mount;    # Send command to mount to get current
DEC
$current_DEC = $response;   # Set current_DEC equal to
response from Gemini
($count_in, $dummy) = $mount_port->read(10); # Clears input register of random
extraneous characters
  # that inexplicably appear from the
gemini mount
$current_DECd = substr($current_DEC, 0, 3); # Split &format current_DEC
degrees  +/-dd
if ( (abs($DECd - $current_DECd)) <= 5 )   # Stop slewing when within 3
degrees of goal
{ print "\nDEC slew is finished\n";
  $response_count = 0;   # Number of characters in Geminis
reply.  Reply depends on command sent.
  $mount_string = 'Telescope has stopped';
  $mount_command = ':Q#';   # Form proper command to quit moving
  &send_to_mount;   # Send command to mount
  ++$dec_check;    # Increment counter to exit FOR loop
 } # end if ( (abs($DECd - $current_DECd) <= 3 )
    } # end $dec_check for loop
  return;
 } # end if ($direction = "dec")

  elsif ($direction eq "ra")   # Monitor the right ascension axis
{ for ($ra_check = 0; $ra_check <= 1; ++$ra_check)
   { $ra_check = 0;
$response_count = 9;   # Number of characters in Geminis
response.  response depends on command sent.
$mount_string = 'Current_RA';   # For debuggin'
$mount_command = ":GR#";   # Form proper command to get the
current RA
&send_to_mount;    # Send command to get RA
$current_RA = $response;   # Set current_RA equal to response
from Gemini
($count_in, $dummy) = $mount_port->read(10); # Clears input register of random
extraneous characters
  # that inexplicably appear from the
gemini mount
```

```
$current_RAh = substr($current_RA, 0, 2);   # Split & format current_RA
hour  HH
if ( $RAh == $current_RAh )
{ print "\nRA slew is finished \n";
  $response_count = 0;   # Number of characters in Geminis
reply.  Reply depends on command sent.
  $mount_string = 'Telescope has stopped';
  $mount_command = ':Q#';   # Form proper command to quit moving
  &send_to_mount;   # Send command to mount
  ++$ra_check;   # Increment counter to exit FOR loop
 } # end if ( $RAh == $current_RAh )
    } # end $ra_checkfor loop
  return;
 } # end elsif ($direction eq "ra")
} ##############################################################################
 # end sub monitor_manual_slew


#--------------------mount park or track--------------------------------#
sub mount { ###########################################################################
 # This subroutine parks the mount OR starts tracking:
 # - If it is sent "park" it will stop mount tracking.
 #   Can slew to any position before or after parking
 #   and mount will not track.  Command ->  &mount(park);
 # - If it receives "track" it will initiate sidereal
 #   tracking.  MOUNT WILL REMAIN STOPPED, AFTER TRACKING
 #   IS INITIATED, UNTIL A SLEW OR MOVE COMMAND IS SENT.
 #   MUST SLEW AFTER TRACKING IS INIATED TO ACTIVATE TRACKING.
 #   Command -> &mount(track);

 my($action) = @_;    # Get action desired.  $action = "park" to stop tracking.
    #    $action = "track" to initiate tracking.
 $error = 0;
 $mount_error = 'none';
# ----- Send and check park commands ----- #
 if ($action eq "park")
 { Park:
   $response_count = 0;  # No reply to this command.
   $mount_string = 'parking sequence initiateed';
   $mount_command = '>135:s#'; # Form proper command to stop tracking.
   &send_to_mount; # send command to mount.
 #----- Check if park command was obeyed ----- #
   $response_count = 5;  # Number of characters in Geminis response.
   $mount_string = 'Tracking Speed Code'; # Code is Gemini's cryptic response to
this command. 131s-> sidereal 135w->no tracking.
   $mount_command = '<130:t#'; # Form proper command to request
tracking speed.
   &send_to_mount; # send command to mount.
   if ($response ne "135w#") # Check if proper command was received.
 {  print "\nPark command not received\n";
 if ( $error < 3 ) # Try to park 3 times if response not
acceptable.
{ ++$error;
  print "Trying again...";
  goto Park; # Try again.
 }# end if ( $error < 3 )
 else # Failed to park 3 times in a row, so shut down.
{ $mount_error = 'error';
  #&tasks_done("*** MOUNT MALFUNCTION! ***");
  #&tasks_done("Parking failed.");
  &shut_down;
 }# end else
  } # end if ($response ne "135w#")
   else  #$response equals "135w#"  Correct parking response received.
```

```
{ $error = 0;
  #&tasks_done("Park successful.");
  print "\nTelescope is parked.\n";
 }# end else
  } #end if ($action eq "park")
# ----- Send and check track commands ----- #
 elsif ($action eq "track")
 { Track:
   $response_count = 0;  # Number of characters in Geminis response.
response depends on command sent.
   $mount_string = 'Sidereal tracking mode set';
   $mount_command = '>131:w#'; # Form proper command to set sidereal
tracking mode.
   &send_to_mount; # IMPORTANT: Must Slew to new field
AFTER sidereal tracking.
 #----- Check if track command was obeyed ----- # # is set!  Mount does not activate tracking
until.
   $response_count = 5;  # Number of characters in Geminis response.
response depends on command sent.
   $mount_string = 'Tracking Speed Code'; # Code is Gemini's cryptic response to
this command. 131s-> sidereal 135w->no tracking.
   $mount_command = '<130:t#'; # Form proper command to request
tracking speed.
   &send_to_mount; # Send command to mount.
   if ($response ne "131s#") # Check if proper command was received.
 { print "\nTrack command not received\n";
if ( $error < 3 ) # Try to track 3 times if response not
acceptable.
    { ++$error;
 print "Trying again...";
 goto Track; # Try again.
}# end if ( $error < 3)
else   # Failed to park 3 times in a row, so shut down.
    { $mount_error = 'error';
 # &tasks_done("*** Mount Malfunction ***");
 # &tasks_done("Track initiation failed.");
 &shut_down;
}# end else
  }# end if ($response ne "135w#")
   else #$response equals "131s#" Correct tracking response received.
{ $error = 0;
  print "\nTelescope tracking initiated.\n";
  # &tasks_done("Telescope tracking initiated.");
 }# end else
  }# end elsif ($action eq "track")
 # ---- Command not recognized ---- #
 else  # $action not equal to "park" AND $action not equal to "track"
 { print "\nCommand Not Recognized.\n";
   $mount_error = 'error';
   # &tasks_done("*** MOUNT MALFUNCTION ***");
   # &tasks_done("Mount Park or track command not recognized.");
   &shut_down;
  }# end else
 #end sub mount
} ################################################################################

sub observe { ###################################################################
#This is the subroutine that tells the camera what to observe.
#
  $roof_status_variable = 0;
  $size = $#inputdata;   #No. of lines in input file
  $ASCII = 96;   #Initialize $variable; the ASCII code 97 = 'a'
  $image_number = 0;   #Initialize; this is the order that this image was taken
```

```
  for ($count=1; $count<=$size; ++$count) { # Read field data from @inputdata
    chomp($inputdata[$count]); # Cut off return (\n) character
    next if ($inputdata[$count] eq "FLAT"); # Skip to next line if this is FLAT cmd.
    if ($inputdata[$count] eq "FILL") {  # Checks to see if a dewar
 &dewar; # refill should occur
 next} # Go to next line in OBS file
    ($RAh,$RAm,$RAs,$DECd,$DECm,$DECs,$field,$filter_goal,$exposure,$num_pics,$except) =
     split(' ',$inputdata[$count]); #Assigns values from each line
    if (substr($DEC,0,1) eq '-') {$sign = -1.0}    # CHeck to see id DEC < 0
      else {$sign = 1.0}

    $filter_name = 'r' if ($filter_goal == 1); # red/continuum filter
    $filter_name = 'h' if ($filter_goal == 2); # H-alpha filter
    $filter_name = 'd' if ($filter_goal == 3); # Dual-band filter
    $filter_name = 's' if ($filter_goal == 4); # SII filter

    for ($pic_num=1; $pic_num<=$num_pics; ++$pic_num) {   #Take multiple images
 &altitude;     # Compute alt-az coordinates
 &time_to_quit;      # Has observing gone beyond end time?
 &HA_15min if ($except ne "EAST");   # Do not observe in East unless instructed

 if ($ALT >= 35.0) {  # OK to observe if altitude >= 35 deg
 @coordinates = ($CRA[0], $CRA[1], $CRA[2], $DECd, $DECm, $DECs);
 if ($inputdata[$count] eq "NOSLEW") {next}  # Don't slew; go to next line in OBS file
 else {&slew(@coordinates)} # Slew telescope
 if ($pic_num == 1) {
   &filter($filter_goal);   # Select which filter to use
 }
 &take_image if ($mount_error ne "error");
  }
  else { # If can't view this object
 &tasks_done("* * * UNABLE TO VIEW THIS FIELD AT THIS TIME * * *");
 &tasks_done("Altitude is less than 35.0 degrees.\n");
 &print_to_file;  print "\n\n"; # Print blank lines to screen.
 $pic_num = $num_pics} # Go to next field to observe

#Make sure that a dewar refill occurs at most every 3 hours
  @now = gmtime; # Current time
  $new_fill = DECIMAL(1.0, $now[2],$now[1],$now[0]);   #...in hours
  $new_fill = Julian_day($now[3],$now[4]+1.0,$now[5]+1900.0,$new_fill,0,0);
  $elapsed_time = $new_fill - $old_fill; # How long since previous fill?
  $elapsed_time = 24.0*$elapsed_time; # Convert time to hours
  if ($elapsed_time >= 3.0) {   # Refill every 3.0 hours
     @home = hor2eq(37.5, 357.0); # Pointing almost at NCP
     &slew(@home);   # Slew to home position
     &dewar;   # Fill dewar
  }
  &check_polaris  # Check after each image to see if the sky is clear/cloudy
    }

    if ($inputdata[$count] eq "NOSLEW") {next} # Don't slew; go to next line in OBS file
    else {
      &tasks_done("Returning to home position...");
      @home = hor2eq(37.5, 357.0); # Pointing almost at NCP
      &slew(@home);     # Slew to home position
      &tasks_done("Home find complete...\n");
    }
    open(LOGFILE, ">>$log_file");  # separate entries in log w/ 2 blank lines
    print (LOGFILE "\n\n");  close (LOGFILE);
  }
  &tasks_done("  * * * END OF NORMAL OBSERVING SESSION * * *\n\n");
} ##############################################################################

sub offsets { ##################################################################
```

```
#A subroutine used by the subroutine startime...
#Compute column and row offsets for reading from file.
#
  my($mn, $dy, $yr) = @_;
  $d = $dy;
  @days_31 = (1, 3, 5, 7, 8, 10, 12);  #Month number of months with 31 days
  for ($entry = 0; $entry <= 6; ++$entry) {
    if (($mn == $days_31[$entry] && $dy == 31) ||   #If month is
  ($mn != $days_31[$entry] && $dy == 30)) {   #not February
$mn = $mn + 1; $dy = 0}
    if ($mn == 2) {   #If month is February
  if (($dy == 29 && $yr % 4 eq 0) || ($dy == 28 && $yr % 4 ne 0)) {
    $mn = $mn + 1; $dy = 0}
    }
  }
  $am_row = $dy + 14;   #row for am
  $pm_row = $d + 13;   #row for pm
  $am_column = 11.0*$mn - 7.0;   #These are the offset values - they tell
  $pm_column = 11.0*$month - 2.0; #how many columns over to start reading
  $mr_column = 11.0*$month - 7.0;
  $ms_column = $pm_column;

  return ($am_row, $pm_row, $am_column, $pm_column, $mr_column, $ms_column);
} ##############################################################################

sub PMIS_start { ###################################################################
# Initialize PMIS for current observing session #

# modified PMIS_start.pl  5/8/02  K.P. Portock
#
# PMIS must be running before activating this program.
# This program communicates with PMIS to do the same functions as
# the PMIS macro START.CMD.  PERL sends the information to PMIS
# that were previously inputted by the user when START.CMD was run.
# Defines current directory in PMIS as the
# directory created by the main program for the current observing session.
# Then it runs c:\slic\newmacs\startnew.cmd (<-location of macro will change in final version)
# which implements the rest of START.CMD.
#
# This program also opens the image window that will be used for the session.
# Takes a junk image (1st image after starting PMIS is always junk -> from NOMINAL
# SETUP AND OBSERVING PROCEDURE
#
# Current directory must be created before executing this subroutine

#  chdir('C:\PMIS');     # Change to PMIS directory
#  system "pmis.exe";     # Launch PMIS application
#  sleep (2);     # Wiat while PMIS catches up
#  chdir($directory);     # Change back to observing directory

  $Client = new Win32::DDE::Client('PMIS', 'CLI'); # Initiates DDE with PMIS, PMIS MUST BE OPEN.
die "Unable to initiate conversation" if $Client->Error;
    # MUST NOT BE CONNECTED prior to this action or PMIS
will freeze.
  sleep(3);     # Wait while PMIS catches up

  $lens = '"58mm"';  # sets $lens to a string = "58mm" <-quotes included in string.
  $ctemp = '"-110.5"'; # sets $ctemp to a string = "-110.5" <-quotes included in string.
# These variables can be set and changed by the main PERL program
accordingly.
  $Client->Execute("vdefine iiii \"$directory\"") || die "DDE execute failed";
# Sets user defined variable in PMIS iiii = current directory, created
by
# &create and sets the path where images will be saved.
```

```
  $Client->Execute("vdefine lens $lens") || die "DDE execute failed";
# Sets user defined variable in PMIS lens = 58mm for fits header
  $Client->Execute("vdefine ctemp $ctemp") || die "DDE execute failed";
# Sets user defined variable in PMIS ctemp = -110.5 for fits header
  $Client->Execute('!c:\slic\newmacs\startnew.cmd') || die "DDE execute failed";
# Runs the macro startnew.cmd  (<-location of macro will change in final
version)
  sleep(2);     # wait while PMIS catches up
  $Client->Execute('menu newimage')     # Opens image window in PMIS at start of observing
session.
 || die "DDE execute failed";   # Default name is Image_1. All images use this
window.
  sleep(2);     # wait while PMIS catches up
  #$Client->Execute("OBS 1"); # Takes a picture, exposure for 1 millisecond.
#<----------------------removed for testing
#  1st Image of the night is always junk, IT IS NOT SAVED.
  #sleep(20); # waits 20s while camera exposes and image is processed
#<-----------removed for testing

} ##############################################################################

sub print_to_file { ############################################################
#Prints the field information to log file. %02d prints 2-digit #'s (ex., 08)
#
  @LST = &decimal_converter($LST/15.0);    #Local sidereal time in HH:MM:SS
  @HA = &decimal_converter($HA/(15.0*$RAD));  #Hour Angle in HH:MM:SS
  @AZ = &decimal_converter($AZ);    #Azimuth in DD:MM:SS
  @ALT = &decimal_converter($ALT);    #Altitude in DD:MM:SS

  #Print to log file
  open (LOGFILE, ">>$log_file"); #Append log file
  print (LOGFILE "\tFIELD NAME: $field\n");
  printf (LOGFILE "\t(RA, DEC) = (%02dh %02dm %02ds, %02dd %02d' %02d\")\n",
 $RA[0],$RA[1],$RA[2],$DEC[0],$DEC[1],$DEC[2]);
  printf (LOGFILE "\t(ALT, AZ) = (%02dd %02d' %02d\", %02dd %02d' %02d\")\n",
 $ALT[0],$ALT[1],$ALT[2],$AZ[0],$AZ[1],$AZ[2]);
  printf (LOGFILE "\tLST = %02d:%02d:%02d \t\t HA = %02d:%02d:%02d\n",
 $LST[0],$LST[1],$LST[2],$HA[0],$HA[1],$HA[2]);
  print (LOGFILE "\tFILTER #: $filter_goal \t\t EXPOSURE TIME: $exposure mins.");
  close (LOGFILE);    #Close log file

  #Print to screen
  print ("\tFIELD NAME: $field\n");
  printf ("\t(RA, DEC) = (%02dh %02dm %02ds, %02dd %02d' %02d\")\n",
 $RA[0],$RA[1],$RA[2],$DEC[0],$DEC[1],$DEC[2]);
  printf ("\t(ALT, AZ) = (%02dd %02d' %02d\", %02dd %02d' %02d\")\n",
 $ALT[0],$ALT[1],$ALT[2],$AZ[0],$AZ[1],$AZ[2]);
  printf ("\tLST = %02d:%02d:%02d \t\t HA = %02d:%02d:%02d\n",
 $LST[0],$LST[1],$LST[2],$HA[0],$HA[1],$HA[2]);
  print ("\tFILTER #: $filter_goal \t\t EXPOSURE TIME: $exposure mins.\n");
} ##############################################################################


sub roof { #####################################################################
# PERL roof control program. Use with roof_v6.bs2.
# This subroutine communicates with the BASIC stamp to open and close the roof.
# $roof_command: 1 to open, 2 to close, 3 to reset counter (PC to Stamp: 'I'm Okay')
# last modified 11/06/02    K.P. Portock
#
  my($roof_command) = @_;   # Get command when roof routine is called
# $Command is set to 1 to open roof, 2 to close roof, 3 to check
# roof status & resets stamp counter.  (Done in main program.)

  $roof_read_time = 600000;
```

```
  &configure_roof_port;         # Change read time

  $command_error = 0;
  $roof_error = "none";
  $roof_error_count = 0;
  $roof_message_count = 0;
#  print "roof_command1 = $roof_command\n";
  if ($roof_command == 3)
     { $roof_read_time = 1000;
&configure_roof_port;     # Change read time
     }

  &roof_status($roof_command); # Check the roof status
} ##############################################################################

sub roof_status { ##############################################################
# Used with the roof subroutine.  This checks the BASIC stamp to determine what the
# roof is doing.
# last modified 11/06/02, 2002 K.P. Portock
#

  my($roof_command) = @_;   # Get command when roof routine is called
# $Command is set to 1 to open roof, 2 to close roof, 3 to check
# roof status & resets stamp counter.  (Done in main program.)

 ROOF_TOP :
#  print "roof_command2 = $roof_command\n";
  $roof_port->write($roof_command);   # send command to stamp
  ($count_in, $echo) = $roof_port->read(1);   # echo 1 digit echo
#  print "\nroof_echo = $echo\n";   # for debugging
# sleep(1);
  ($count_in, $check_roof) = $roof_port->read(1); # get roof stamp's reply, waits 10 minutes
     print "\ncheck_roof = $check_roof\n";   # 1 digits reply from Stamp

   if (length($check_roof) < 1 )   # If a 1 digit reply is not read then there is
an error
      { if ($roof_command == 3)    # If roof command is a status check
   { print "\nRoof Stamp did not receive command.";  # write to screen
     ++$command_error;   # increment error counter
     if ($command_error >=5)   # if error occurred 5 times then shut_down
{ &tasks_done("*** ROOF MALFUNCTION!!! Stamp could not receive command from
PERL. ***");
  $roof_error = "error";   # Raise error flag
  &shut_down;   # Shut down
 } # end if ($command_error >=5)
     else   # If error occured less than 5 times then try
again
{ print "  Trying again...";
  goto ROOF_TOP;   # Go back to resend command to the stamp
 } # end else error is less than 5
     } # end if ($roof_command = 3)
 else   # Roof command is open or close
    { &tasks_done("*** ROOF MALFUNCTION!!!  No reply from Stamp after 10 minutes. ***");
      $roof_error = "error";   # Raise error flag
      &shut_down;   # Shut down
     } # end else command is open or close
       } # end if (length($check_roof) < 1 )

   if ($check_roof == 1)    # receive a 1, roof is opening
 { if ( $roof_message_count < 1 )    # print message only once
 {  &tasks_done("The roof is opening...");
    ++$roof_message_count;
  } # end if ( $roof_message_count < 1 )
   $roof_error_count = 0;    # reset error counter
```

```
  } # end if ($check_roof == 1)
   elsif ($check_roof == 2)    # receive a 2, roof has stopped opening
 { if ($roof_status_variable < 1)    # print message only once
 {  &tasks_done("The roof is open.\n");
    $roof_error_count = 0;    # reset error counter
    ++$roof_status_variable;    # increment counter
  } # end if ($roof_status_variable < 1)
   return;
  } # end elsif ($check_roof == 2)
   elsif ($check_roof == 3)    # receive a 3, roof is closing
 { if ( $roof_message_count < 1 )    # print message only once
  { &tasks_done("The roof is closing...");
 ++$roof_message_count;
} # end elsif ($check_roof == 3)
    $roof_error_count = 0;    # reset error counter
  } # end elsif ($check_roof == 3)
   elsif ($check_roof == 4)    # receive a 4, roof has stopped closing
 { if ($roof_status_variable < 1)    # print message only once
  { &tasks_done("The roof is closed.");
 $roof_error_count = 0; # reset error counter
 ++$roof_status_variable; # increment counter
} # end if ($roof_status_variable < 1)
   return;
  } # end elsif ($check_roof == 4)
   elsif ($check_roof == 5)    # receive a 5, the stamp has timed out
 { &tasks_done("*** THE ROOF TIMED OUT ***");
   $roof_error = "error";
   &shut_down;
  } # end elsif ($check_roof == 5)
   elsif ($check_roof == 6)    # receive a 6, the roof is in between the
limit switches
 { &tasks_done("The roof is in transit...");
   $roof_error_count = 0;    # reset error counter
  } # end elsif ($check_roof == 0)
   elsif ($check_roof == 7 )    # recieve a 7, power has gone out and roof is
automatically closing
{ &tasks_done ("*** POWER FAILURE DETECTED ***"); # so shut_down
  $roof_error = "error";
  &shut_down;
} # end elseif ($check_roof ==7)
   elsif ( $check_roof == 8 )    # receive an 8, roof failed to close in correct time
{ &tasks_done ("*** ROOF MALFUNCTION.  Stamp indicates that the roof has failed. ***");
  &tasks_done ("*** PHONE HOME ***");
  $roof_error = "error";
  &shut_down;
} # end elsif ($check_roof == 8)

} ##############################################################################

sub save_image { ###############################################################
# adapted from savefits.pl    5/16/02     K.P. Portock
# This program communicates with PMIS to do the same functions as
# the PMIS macro FITS.CMD.  PERL sends the information to PMIS
# that were previously inputted by the user when FITS.CMD was run.
# Then it runs c:\slic\newmacs\fitsnew.cmd which implements
# the rest of FITS.CMD.
# This routine is used to save all images including correction images.
# Run this immediately after the take_image routine to save the current image.
#
# Outputs each as a 16-bit FITS file with useful keywords.
# Image values are shifted down by 32768, so high intensity pixels
# will have higher integer values than low intensity pixels (no wrap-around
# along intensity axis).  FITS values BZERO and BSCALE will make sure
# that correct pixel values (between 0 and 2**16) result when the FITS file
```

```
# is read into AIPS or IRAF.
#
  $Client->Execute("vdefine object \"$field\"") || die "DDE execute failed";
    # sets user defined variable OBJECT in PMIS.
  $Client->Execute("vdefine ifilter $filter") || die "DDE execute failed";
    # sets user defined variable IFILTER in PMIS.
  $Client->Execute("vdefine type \"$image_type\"") || die "DDE execute failed";
    # sets user defined variable TYPE in PMIS.
  $Client->Execute("vdefine fname \"$file_name\"") || die "DDE execute failed";
    # sets user defined variable FNAME in PMIS.
  $Client->Execute('!c:\slic\newmacs\fitsnew.cmd') || die "DDE execute failed";
    # runs the macro fitsnew.cmd -> processes and saves current image as
a
    # fits file.  See documentation above.
  sleep(2);     # waits while PMIS saves the image and catches up
  open (LOGFILE, ">>$log_file");
  print (LOGFILE "\tImage saved to: $file_name.\n");
  print ("\tImage saved to: $file_name.\n");
  close (LOGFILE);
} ###########################################################################

# ------------------- Send and check DEC command ------------------------- #

sub SEND_DEC { ######################################################################
   $response_count = 1;  # Number of characters in Geminis response. response
depends on command sent.
   $mount_string = 'DEC sent'; # Declination of the field, DEC must be in
+/-dd:mm:ss format
   $mount_command = ":Sd$DECd:$DECm:$DECs#"; # Form proper command to set new DEC
   &send_to_mount;
   if ($response != 1) # Check for correct mount response.
 { ++$error; # Response of 1 means valid command,
print "\nError sending DEC command."; # see gemini level2 command set for details.
if ($error <= 3) # If wrong response the try again
   { print "  Trying again..."; # Try 3 times then shut down
($count_in, $dummy) = $mount_port->read(10); # Clear input register
&SEND_DEC;
    }# end if ($error <= 3)
else
   { $mount_error = 'error'; # Messed up three times, shut down.
&tasks_done("*** MOUNT MALFUNCTION ***");
&tasks_done("*** Error sending DEC command to Gemini mount***");
&shut_down;
    }# end else
  } # end if ($response != 1)
   else { $error = 0;}
} ###########################################################################
# end sub SEND_DEC

#-------------------------------------------------------------------------#
# ----- Send command to name the selected object ----- #
sub send_name { #####################################################################
   $response_count = 0;  # Number of characters in Geminis response. response
depends on command sent.
   $mount_string = 'Name sent';  # Debugging
   $mount_command = ":ON$field#"; # Tell Gemini Mount the name of the selected
object
   &send_to_mount;
} ###########################################################################
# end sub send_name

#-------------------------------------------------------------------------#
sub SEND_RA { #######################################################################
  # RA & DEC given as: $RAh:$RAm:$RAs, $DECd:$DECm:$DECs
```

```
  # ----- Send and check RA command ----- #
   $response_count = 1;  # Number of characters in Geminis response. response
depends on command sent.
   $mount_string = 'RA sent'; # Right Ascension of the field, RA Must be in hh:mm:ss
format
   $mount_command = ":Sr$RAh:$RAm:$RAs#"; # Form proper command to set the new RA
   &send_to_mount;
   if ($response != 1) # Check for correct mount response.
 { ++$error; # Response of 1 means valid command,
print "\nError sending RA command."; # see gemini level2 command set for details.
if ($error <= 3) # If wrong response the try again
   { print "  Trying again..."; # Try 3 times then shut down
($count_in, $dummy) = $mount_port->read(10); # Clear input register
&SEND_RA;
    }# end if ($error <= 3)
else
   { $mount_error = 'error'; # Messed up three times, shut down.
&tasks_done("*** MOUNT MALFUNCTION ***");
&tasks_done("Error sending RA command to Gemini mount");
&shut_down;
    }# end else
  }# end if ($response != 1)
   else { $error = 0;}
} ###########################################################################
# end sub SEND_RA

#---------------------- Send Command to Gemini Mount ----------------------#
# universal subroutine that sends gemini mount commands
# Must set $command to valid form.
# Set $string to desired output.
# Set $response_count to number of characters in expected response
sub send_to_mount { ###############################################################

$mount_port->write($mount_command);     # Send command.
($count_in, $response) = $mount_port->read($response_count);   # Read Geminis response,
see Gemini level2 command set
# print "$mount_string: $response\n";     # Leave in when debugging.
} ###########################################################################
#Write to file that telescope is moving.


sub shut_down { ###################################################################
#Performs shut down procedures - powers of camera and AE-25, parks 'scope,
#closes roof, closes observing directory, turns of Polaris monitor
#

  &tasks_done("Sytem shut down in progress....\n");
# --- RETURN FOCUS TO INFINITY --- #
  if ($focus_error ne "error")   # Check for focusing error flag
{ &focus("0000");    # Return focus to home (See 'sub focus for
details)
 } # end if ($focus_error ne "error")
# --- PARK TELESCOPE --- #
  if ($mount_error ne "error") # Check for mount error flag
{ &tasks_done("Parking telescope."); # Writes task completed to file
 @park = hor2eq(37.5, 357.0); # Coords. for rest position
 &slew(@park);    # Slew telescope to park position.
 &mount(park); # Halt tracking.
 &tasks_done("Telescope park complete.\n"); # Writes task completed to file
 } # end if ($mount_error ne "error")

# --- CLOSE ROOF --- #
   if ($roof_error ne "error")  # Check for roof error flag, close roof if none
  { $roof_status_variable = 0;  # Show roof message only once
```

```
  &roof(2);  # Close roof
} # end if ($roof_error ne "error")

# --- CLOSE PMIS --- #
  &tasks_done("Closing PMIS applications.");   # Writes task completed to file
  $Client->Execute('iremove "Image_1" ') || die "DDE execute failed"; #closes PMIS window called
"Image_1"
  sleep(2);   # Wait while PMIS catches up
  $Client->Disconnect;   # Disconnect PERLs DDE communication with PMIS
  # Must be disconnected BEFORE trying to
connect or reconnect. (see above)
  sleep(2);   # Wait while PMIS catches up
  &tasks_done("PMIS shut-down complete.\n");   # Writes task completed to file

# --- REMOVE SERIAL PORTS --- #
#  undef $ffd_port;   # Focus, filter and dewar port
#  undef $roof_port;  #roof port
#  undef $mount_port;  #mount port

  &copy_files;  #Copy LOG, OBS and images to back-up directory
  print ("\nShut-down complete... GOOD NIGHT~!\n"); #All tasks are done!

#This block causes the program to wait until the next day and then returns the
#program to the beginning of the MAIN block.  $JD_new_day is defined in site_data.
#
  @now = gmtime;
  $now = Julian_day($now[3], $now[4]+1, $now[5]+1900, $now[2], $now[1], $now[0]);
  $print_status = $now;
  $new_obs_day = int($now + 1) - 2440587.5;  # How long since epoch (Jan. 1, 1970, 00:00 UT)
  $new_obs_day = $new_obs_day*60*60*24;
  $new_obs_day = scalar(gmtime($new_obs_day));

  do {
    @now = gmtime;  $JD_now =
  Julian_day($now[3], $now[4]+1, $now[5]+1900, $now[2], $now[1], $now[0]);
    $status = $JD_now - $print_status;
    $roof_status_variable = 1;
    if ($roof_error ne "error") {
      sleep(10);
      &roof(3)}     # Ping roof to get status. Roof will timeout after 20 minutes
    if ($status > 1.0/96.0 || $status == 0) {  #Print status every 15 minutes
  print "\nCurrently it is ", scalar(gmtime), " UT.\n";
  print "The new observing day begins on $new_obs_day UT.\n";
  $print_status = $JD_now; sleep(5)}
  } while ($JD_now < $JD_new_day);     #Wait until noon on new day to try again
 exec 'perl C:\SLIC\slictest.pl';
} ################################################################################

sub site_data { ###################################################################
#This subroutine contains some basic information about the SLIC camera
#and Vorticella Lab.  it also contains some constants used in several
#calculations throughout the program.
#

  $PI = 4.0*atan2(1.0, 1.0); #Defines pi = 3.14159...
  $RAD = $PI/180.0;      #Constant used to Convert betw. degrees & radians
  $LAT = DECIMAL(1.0, 37, 19, 48);    #Lat. of SLIC (37d 19m 48s N)
  $LAT = $LAT*$RAD;    #Latitude in radians
  $LNG = DECIMAL(-1.0, 80, 33, 30);   #Long. of SLIC (80d 33m 30s W)
  $LNG = $LNG*$RAD;    #Longitude in radians

#This is the date and time that the session begins.  This value will be used by a
#loop in the shut_down subroutine to tell the PC to begin a new observing day at
#noon (localtime) the next day.
```

```
  @new_day = localtime;
  $new_d = $new_day[3]; $new_mn = $new_day[4] + 1;  $new_yr = $new_day[5] + 1900;
  $new_h = $new_day[2]; $new_m = $new_day[1]; $new_s = $new_day[0];
  $JD_new_day = Julian_day($new_d, $new_mn, $new_yr, $new_h, $new_m, $new_s);
  if ($JD_new_day - int($JD_new_day) >= 0.5) {$JD_new_day = int($JD_new_day + 0.5)}
    else {$JD_new_day = int($JD_new_day) + 1.0}

  print ("\n\t The Virginia Tech\n");  #Greetings!
  print ("\t\tSpectral-line Imaging Camera\n");
  print ("\t * * * Control Program * * *\n\n");
} ##############################################################################

sub slew { ######################################################################
# contains the slew routine, and calls the slew checking procedures
  my(@c)=@_;
  $centering_error2 = 0;  # Initialize variable
  &mount(track);

  SLEW_TOP:
   if ($roof_error ne "error") {&roof(3)} # Ping roof to get status. Roof will timeout
after 20 minutes
   &slew_telescope(@c);   # Send slewing commands
   &check_slew_reply;  # Check geminis reply
   if ($centering_error eq "error")
 { $response_count = 0;   # Number of characters in Geminis response.
$mount_string = 'Quit Slew';  # For debuggin'
$mount_command = ":Q#";   # Form proper command to quit moving
&send_to_mount;   # Send command to mount to get current DEC
++$centering_error2;
if ( $centering_error2 >= 3 )  # If tried 3 or more times then shut down
   { $mount_error = 'error';
&shut_down;
    } # end if ( $centering_error2 >= 3 )
goto SLEW_TOP;  # Try slew again
  } # end if ($centering_error eq "error")
} ##############################################################################
# end sub slew

#---------------------- The Slew Monitor ---------------#
sub slew_monitor { ##############################################################
# This subroutine montitors the progress of mount as it slews to new coordinates
# It loops and periodically checks the RA and DEC of the mount and compares it with the
# previous check.  If the RA and DEC haven't changed for 3 consecutive checks then it assumes
# the mount is finished slewing.  It monitors the speed of the slew. When the mount
decelerates
# to centering speed a timer starts.  If the centering time exceeds 5 minutes the routine
# is aborted.

# --- Monitor RA & DEC Slew --- #
  $last_RA = 'none';    # Initialize variables
  $last_DEC = 'none';
  $center_count = 0;
  $centering_error = 'none';

  print "\nSlewing...";
  for ($slew_count = 0; $slew_count <= 2; ++$slew_count)  # Loop until 3 consecutive values of
RA & DEC are equal
    { sleep(1);     # Wait 1 second
 print ".";

 $response_count = 9;    # Number of characters in Geminis response.
response depends on command sent.
 $mount_string = 'Current_RA';    # For debuggin'
 $mount_command = ":GR#";    # Form proper command to get the
```

```
current RA
 &send_to_mount;    # Send command to get RA
 $current_RA = $response;    # Set current_RA equal to response
from Gemini

 $response_count = 10;    # Number of characters in Geminis response.
Response depends on command sent.
 $mount_string = 'Current_DEC';     # For debuggin'
 $mount_command = ":GD#";    # Form proper command to get the
current DEC
 &send_to_mount;    # Send command to mount to get
current DEC
 $current_DEC = $response;    # Set current_DEC equal to response from
Gemini

 ($count_in, $dummy) = $mount_port->read(10); # Clears input register of random
extraneous characters
   # that inexplicably appear from the gemini
mount
 #print "In FOR loop   slew_count = $slew_count\n\n"; # For debuggin'
 while ( ($current_RA ne $last_RA) || ($current_DEC ne $last_DEC) ) # Loop until two
consecutive values of RA &DEC are equal
  { $slew_count = 0;    # Consecutive values are not equal, reset
$slew_count
    print ".";
    $last_RA = $current_RA;    # Shift variable into $last_RA
    $last_DEC = $current_DEC;    # Shift variable into $ last_DEC

    $response_count = 9;    # Number of characters in Geminis response.
    $mount_string = 'Current_RA';    # For debuggin'
    $mount_command = ":GR#";    # Form proper command to get the current RA
    &send_to_mount;    # Send command to get RA
    $current_RA = $response;    # Set current_RA equal to response from
Gemini

    $response_count = 10;    # Number of characters in Geminis response.
    $mount_string = 'Current_DEC';    # For debuggin'
    $mount_command = ":GD#";    # Form proper command to get the current DEC
    &send_to_mount;    # Send command to mount to get current DEC
    $current_DEC = $response;    # Set current_DEC equal to response from Gemini
    ($count_in, $dummy) = $mount_port->read(10); # Clears input register of random
extraneous characters

    $response_count = 1;    # Number of characters in Geminis response.
    $mount_string = 'mount_speed';    # For debuggin'
    $mount_command = ":Gv#";    # Form proper command to get the mount speed
    &send_to_mount;    # Send command to mount to get mount speed
    $mount_speed = $response;    # set $mount_speed equal to response from gemini

    if ($mount_speed eq "C")    # If mount is centering start timer
  { if ($center_count == 0 ){ print "\nCentering..."; }  # print centering once
 ++$center_count;    # Increment centering timer
 if ( $center_count >= 600 )    # If centering for 5 minutes then exit slew
routine
   { &tasks_done("*** MOUNT MALFUNCTION***");
&tasks_done("Trying again...");
$centering_error = "error";
return;
   } # end if ( ($delta_RA_f > $delta_RA_i) || ($delta_DEC_f > $delta_DEC_i) )
} # end if ($mount_speed eq "C")

    #print "\nIn WHILE loop  slew_count = $slew_count\n\n";
   } # end while ( $current_RA ne $last_RA )
} # end for ($slew_count = 0; $slew_count <= 3; ++$slew_count)
```

```
  $current_RAh = substr($current_RA, 0, 2);    # Split & format current_RA hour  HH
  $current_RAm = substr($current_RA, 3, 2);    # Split & format current_RA minutes MM
  $current_RAs = substr($current_RA, 6, 2);    # Split & format current_RA seconds SS
  $current_DECd = substr($current_DEC, 0, 3);    # Split &format current_DEC degrees +/-dd
  $current_DECm = substr($current_DEC, -6, 2);    # Split & format current_DEC minutes  mm
  $current_DECs = substr($current_DEC, -3, 2);    # Split & format current_DEC seconds  ss
  ($count_in, $dummy) = $mount_port->read(10);    # Clears input register
  print " Done!\n";
  sleep(1); # For debugging
} ##############################################################################
# end sub slew_monitor

sub slew_telescope { ###########################################################
#Moves the camera to the desired coordinates.
#
  my(@position) = @_;

  $RA = &DECIMAL(1.0, $position[0], $position[1], $position[2]);     # Gives RA in hours
  $CRA = $RA - $delta_HA;   # RA corrected for time diff. betw. Gemini and PC
  $CRA = time_range($CRA);   # Make hours >= 0 and < 24
  @CRA = decimal_converter($CRA); # Corrected RA - hh:mm:ss
  $CRA = $CRA*15.0;   # Gives corrected RA in degrees
  $position[0] = $CRA[0]; $position[1] = $CRA[1]; $position[2] = $CRA[2];
       # These are the corrected RA coordinates

  &format_coord(@position); # Format RA and DEC coordinates for checking routine

#Write to file that telescope is moving.
  &tasks_done("Slewing telescope to ($RAh:$RAm:$RAs, $DECd:$DECm:$DECs).\n");

# COMMAND TO GEMINI MOUNT HERE!!!!
   $mount_error = 'none';     # Initialize variable
   $error = 0;     # Initialize variable

  # SEND_SLEW_COMMANDS:
    &SEND_RA; # Send RA coord. to Gemini
    &SEND_DEC; # Send DEC coord. to Gemini
    &send_name;  # Send name of field to gemini

    $response_count = 1;  # For this command, only get 1st character of the
response to
    $mount_string = 'Sending slew command';  # determine what the response is in the
check_slew_reply.
    $mount_command = ':MS#';  # Tell gemini Mount to Slew to the selected
object
    &send_to_mount;

} ##############################################################################

sub startime { #################################################################
#This subroutine reads the tables of moonrise-set and AM/PM twilight times
#produced by http://aa.usno.navy.mil/data/docs/RS_OneYear.html. These tables
#should be saved as "moon.htm" and "twilight.htm," respectively.  Choose the
#option, "Save as type: WEB PAGE, HTML ONLY (*.htm, *.html)' in IE.

#Reads the twilight times from 'twilight.htm' into an array called twilight
#This file should be stored in C:\SLIC
  $twi_table = 'C:\SLIC\twilight.htm';   #Name/location of observing file
  if (-s $twi_table) {   #Check file size
    open(TWILIGHT, $twi_table);  #open file if it exists
    @twilight = <TWILIGHT>; #Creates an array of observing data
    close (TWILIGHT)}  #Closes input file
  else {
```

```
    print "\n*** UNABLE TO LOCATE TWILIGHT TABLES, $moon_table. ***\n";
    &shut_down}

#Reads the moon times from 'moon.htm' into an array called moon.
#This file should be stored in C:\SLIC
  $moon_table = 'C:\SLIC\moon.htm';  #Name/location of observing file
  if (-s $moon_table) {   #Check file size
    open(LUNAR, $moon_table);  #open file if it exists
    @moon = <LUNAR>;  #Creates an array of observing data
    close (LUNAR)}  #Closes input file
  else {
    print "\n*** UNABLE TO LOCATE MOON-TIMES TABLE, $moon_table. ***\n";
    &shut_down}

#Get column and row data
  @off = offsets($month, $day, $year4);

#Read start of amtwilight for tomorrow morning
  $amtwilight = substr($twilight[$off[0]], $off[2], 4);

#Read end of pmtwilight for current evening
  $pmtwilight = substr($twilight[$off[1]], $off[3], 4);

#Read moonrise time for current day
  $moonrise = substr($moon[$off[1]], $off[4], 4);
  $rise_day = $day;

#Read moonset time for current day
  $moonset = substr($moon[$off[1]], $off[5], 4);
  $set_day = $day;

  if ($moonrise eq '') {   #The moon does not rise today
    ($row, $column) = last_day($month, $year4, $off[1], $off[4]);
  #Tells which row & column of moon.htm to read
    $moonrise = substr($moon[$row], $column, 4);
    $rise_day = $rise_day + 1;
    goto JUL_DAY;
    }

  if ($moonset eq '    ') {  #The moon does not set today
    ($row, $column) = last_day($month, $year4, $off[1], $off[5]);
  #Tells which row & column of moon.htm to read
    $moonset = substr($moon[$row], $column, 4);
    $set_day = $set_day + 1; $rise_day = $rise_day + 1;
    goto JUL_DAY;
    }

#If moon rises this evening and sets the next morning, $moonset should be
#tomorrow's moonset time
  if ($moonrise > $moonset && $moonrise < $pmtwilight) {
    ($row, $column) = last_day($month, $year4, $off[1], $off[5]);
    $moonset = substr($moon[$row], $column, 4);
    $set_day = $set_day + 1; $rise_day = $rise_day + 1;
    goto JUL_DAY;
    }

#Moon rises before AM twilight begins
  if ($amtwilight > $moonrise) {
    ($row, $column) = last_day($month, $year4, $off[1], $off[4]);
  #Check to see if today is the last day of the month. If so, may
  #need to read from next column of data in rise/set table.
    $moonrise = substr($moon[$row], $column, 4);  #Moonrise time
    $rise_day = $day + 1 if ($message != 1)}   #Used to compute JD of moonrise
  elsif ($amtwilight < $moonrise && $pmtwilight > $moonrise){
```

```
    #Moonrise occurs after am twilight but before pm twilight
    ($row, $column) = last_day($month, $year4, $off[1], $off[4]);
  #Check to see if today is the last day of the month. If so, may
  #need to read from next column of data in rise/set table.
    $moonrise = substr($moon[$row], $column, 4);  #Moonrise time
    $rise_day = $rise_day + 1}  #Used to compute JD of moonrise

JUL_DAY:
#Change times to UT by adding 5 hours
  $pmtwilight = $pmtwilight + 500; $amtwilight = $amtwilight + 500;
  $moonrise = $moonrise + 500; $moonset = $moonset + 500;

#Take the times read above and convert to hours
  @amtwilight = XXMMSS(1.0, 100.0*$amtwilight); $am = DECIMAL(1.0, @amtwilight);
  @pmtwilight = XXMMSS(1.0, 100.0*$pmtwilight); $pm = DECIMAL(1.0, @pmtwilight);
  @moonrise = XXMMSS(1.0, 100.0*$moonrise); $rise = DECIMAL(1.0, @moonrise);
  @moonset = XXMMSS(1.0, 100.0*$moonset); $set = DECIMAL(1.0, @moonset);

#Give the Julian day for each of these events
  $pm_JD = Julian_day($day, $month, $year4) + $pm/24.0;
  $am_JD = Julian_day($day+1, $month, $year4) + $am/24.0;
  $rise_JD = Julian_day($rise_day, $month, $year4) + $rise/24.0;
  $set_JD = Julian_day($set_day, $month, $year4) + $set/24.0;

#The following section uses twilight and moonrise/-set times to determine when
#observing should begin (i.e., when initial dewar fill should start).
#
#Starting time is moonset if moonset occurs before pm twilight. Else, the
#start time is the end of pm twilight.
#Observations end at the beginning of am twilight if am twilight occurs prior
#to moonrise. Otherwise, moonrise marks the end of observing.
#
#Also, if there is less than 1 hour between the starting and ending times of
#observing, no observing will occur.

  if ($pm_JD < $set_JD) { #JD of start
    $start = $set_JD;  @starting = @moonset;
    $start_tag = "MOONSET"} #Tells user that this time is moonset
  else {
    $start = $pm_JD;  @starting = @pmtwilight;
    $start_tag = "PM TWILIGHT"}  #Tells user that this time is pm twilight

  if ($am_JD < $rise_JD) { #JD of stop
    $stop = $am_JD;  @ending = @amtwilight;
    $end_tag = "AM TWILIGHT"} #Tells user that this time is am twilight
  else {
    $stop = $rise_JD;  @ending = @moonrise;
    $end_tag = "MOONRISE"} #Tells user that this time is moonrise
  $duration = 24.0*($stop - $start); #How long can we observe, in hours
  @duration = decimal_converter($duration); #Convert to HH:MM:SS
} ###############################################################################

sub take_bias { ###################################################################
#  take_bias.pl K.P. Portock  5/16/02
# Edited take_image program to take 10 bias images.
# Uses macro startbias.cmd and savebias.cmd.
# startbias.cmd and savebias.cmd are lines copied from
# the original bias10.cmd.  Exposure time = 0 seconds.
#
#Must first turn off all lights, close doors and roof, before taking bias images.

  $exposure_time = 0;  # must be in milliseconds
  $image_type = 'bias';
  $field = 'bias';
```

```
  $filter = 0;
  $wait_time = $exposure_time/1000+ 20;
  $hours =  (($exposure_time/1000)+ 20)/3600;
  @time_lapse = &decimal_converter($hours);  # time that PERL program waits as image is
exposed and processed by PMIS
 # This time is the exposure time (converted to
seconds) plus 20 seconds
  $num_pics = 0;

  print "\nCapturing bias images...\n";
  open (LOGFILE, ">>$log_file");  #Append log
  print (LOGFILE "\n");
  close(LOGFILE);
  while( ++$num_pics <= 10 )  # Loops 10 times to take 10 bias images
 {
   $Client->Execute('bias');  # Sends command to PMIS to take bias image
   &tasks_done("Capturing bias image (0 sec. exposure).\n");
   printf "\nWaiting %02d min %02d sec for exposure & image transfer...\n",
   $time_lapse[1], $time_lapse[2];  # Displays PERLs wait time for exposure processing
   substr($num_pics,0,0) = '0' x (2-length($num_pics));
   $file_name = 'bias'.$num_pics.'.fit'; # Name of file is biasxx.fit. xx: 2 digit image
number
   sleep($wait_time);  # Waits while camera exposes and image is
processed
   &save_image;   # Call saving routine
  }
} ###########################################################################

sub take_image ##################################################################
#Routine to expose an image     K.P. Portock   5/16/02
#Before running this routine the image parameters must be set to desired values
#The required parameters are:
# $exposure   -> Time of exposure.  In minutes
# $file_name  -> Name the file will be saved under
#     -> $file_name = '"file"' -> file.fit
# $field     ->Info about image; documentation in header of fits file
# $filter -> Filter used to take image; change to desired filter &
# documentation in fits header
# $image_type -> Object, bias, flat, other;  documentation in fits header
# $num_pics   -> Number of exposures that are desired

# Exposes n images and saves them by calling &save_image
# Images are given names = $file_name$i where $i is the number
# of the image in the sequence 1-n.  They are saved in fits format
# K.P. Portock 5/17/02
{
# Give each image a unique name, using the format xxzaaa##.fit, as outlined in
# SLIC Tutorial.doc

  &roof(3); # Ping roof to get status. Roof will timeout after 20 minutes

  substr($image_number, 0, 0) = '0'x(2 - length($image_number)) if ($ASCII < 97);
  $image_type = 'object'; # Define values for fits header info
  if ($image_number < 100 && $ASCII < 97)
  { $file_name = $image_number.$filter_name.$field.'.fit'}
  else
  { if ($image_number > 9)
    { $ASCII = $ASCII + 1;
 $image_number = 0;
} # end if ($image_number > 9)
 $character = chr($ASCII);
 $file_name = $character.$image_number.$filter_name.$field.'.fit';
} # end else
```

```
  ++$image_number; # Increment image number count
  &tasks_done("Capturing image $pic_num/$num_pics of $field.\n"); # Print status message
to screen
  &print_to_file;

  $exposure_time = 60000.0*$exposure;   # Exposure time in milliseconds
  $wait_time = $exposure_time/1000+ 20;    # Exposure time in seconds +20 seconds
  $hours =  (($exposure_time/1000)+ 20)/3600;
  @time_lapse = &decimal_converter($hours);   # Time that PERL program waits as image is
exposed and processed by PMIS
# This time is the exposure time (converted to
seconds) plus 20 seconds
  $Client->Execute("OBS $exposure_time");   # Tell PMIS to expose for $exposure_time
milliseconds
 # takes a picture, exposure for $exposure_time milliseconds
 # die "DDE execute failed"; is left out leaving this line in the code
 # causes an error: "DDE execute failed at PMIS_COM_test.pl line##."
   printf "Waiting %02d min %02d sec for exposure & image transfer...",$time_lapse[1],
$time_lapse[2];
# Displays PERLs wait time for exposure
processing
  sleep($wait_time);   # Waits while camera exposes and image is
processed
  &tasks_done("Image exposure and transfer complete.\n");
  &save_image;   # Calls routine to save the image
} ###########################################################################

sub tasks_done { ###########################################################
#Writes the tasks completed and time when done to the screen and to the log file.
#
  my($comment) = @_;
  $task = "\n".scalar(gmtime)." UT- ".$comment; #."\n";
  open (LOGFILE, ">>$log_file"); #Append log file
  print (LOGFILE $task); #Write task status to log file
  print $task; #Write task status to screen
  close (LOGFILE); #Close log file
} ###########################################################################

sub time_range { ###########################################################
#A routine used by several other subroutines
#
#Make sure that times are between 0 - 24 hours
#
  my($clock) = @_;
  $clock = $clock - 24.0 while ($clock >= 24.0);
  $clock = $clock + 24.0 while ($clock < 0.0);
  return $clock;
} ###########################################################################

sub time_to_quit { #########################################################
#Compares current time to the end-time as determined in the startime subroutine.
#Closes down observing session if we've gone beyond this time.
#
  @now = gmtime;     #Current Universal Time
  $now =     #Julian Day for current time
    Julian_day($now[3], $now[4]+1, $now[5]+1900, $now[2], $now[1], $now[0])/24.0;
  if ($now > $stop) {     #Close program if go beyond stop time
    open (LOGFILE, ">>$log_file");  #Append log
    print (LOGFILE "\n");     #Print blank line to log
    close (LOGFILE);     #Close log file
    &tasks_done("OBSERVATIONS HAVE EXCEEDED OBSERVING WINDOW.\n");
    &shut_down}
} ###########################################################################
```

```
sub UT2LST { ###########################################################################
#Converts universal times to local sidereal times.  Computations were adapted from
#the book, _Practical Astronomy With Your Calculator, 3rd Edition_, by Peter
#Duffett-Smith, pgs. 17 & 20
#
   my($UT, $JD_day, $JD_mon, $JD_yr) = @_;
   $JD = Julian_day($JD_day, $JD_mon, $JD_yr, 12, 0, 0); #Julian Date for noon today

   $S = $JD - 2451545.0;  #Julian days since 12:00 UT 1/1/2000
   $T = $S/36525.0;   #Julian centuries since 12:00 UT 1/1/2000
   $T_0 = 6.697374558 + (2400.051336 + 2.5862E-5*$T)*$T;
   $T_0 = time_range($T_0);
   $time = 1.002737909*$UT;
   $time = $time + $T_0;
   $time = time_range($time);
   $time = $time + $LNG/(15.0*$RAD);  #Compute LST
   return time_range($time);  #Return value of LST, 0 - 24
} ###########################################################################################

sub wait_to_start { ###################################################################
#Wait until time indicated by startime subroutine to start observing
#
  $roof_status_variable = 1; #Print some roof status messages only once
  if ($duration < 1.0) { #Shut down if observing time is less than one hour
    &tasks_done("* * * LUNAR PHASE NOT SUITABLE FOR OBSERVING. * * *");
    &tasks_done("Observing time is less than 1 hour.\n");
    &shut_down } #Tells system to shut_down and wait until the new day
  else { #Or, wait until it is time to begin
    $start_dew = time_range($starting[0]-2);  #Make start time betwe. 0 - 24
    $ending[0] = time_range($ending[0]);  #Make end time betwe. 0 - 24
    @now = gmtime; $print_status =   #Julian date for current time
  Julian_day($now[3], $now[4]+1, $now[5]+1900.0, $now[2], $now[1], $now[0]);

    do {
 @now = gmtime;  $JD_now =
  Julian_day($now[3], $now[4]+1, $now[5]+1900.0, $now[2], $now[1], $now[0]);
 $status = $JD_now - $print_status;
 if ($status > 1.0/96.0 || $status == 0) {  #Print status every 15 minutes
print "\nThe current time is ", scalar(gmtime), ".";
print "\nInitial dewar fill begins 2 hours before $start_tag ";
printf "at %02dh %02dm UT.", $start_dew, $starting[1];
printf "\nObservations should end by $end_tag (%02dh %02dm UT).\n",
  $ending[0], $ending[1];
$print_status = $JD_now; sleep(5);
 }
sleep(10);
&roof(3); # Ping roof to get status. Roof will timeout after 20 minutes
    } while ($JD_now < $start - 1/12);   #Wait until 2 hrs before $start
  }
} ###########################################################################################

sub XXMMSS { ##########################################################################
#This subroutine chops a number, XXMMSS, and puts it into the format
#XX:MM:SS
#

  my($sn, $DATA, @UNITS) = @_;
  $DATA = abs($DATA);
  $UNITS[0] = int($DATA/10000.0);   #Gives whole number of degrees/hours
  $UNITS[1] = int($DATA/100.0);    #Gives whole number of minutes
  $UNITS[1] = $UNITS[1] % 100.0;
  $UNITS[2] = $DATA % 100.0;   #Gives seconds
  if ($UNITS[0] ne 0) {
    $UNITS[0] = $sn*$UNITS[0]}
```

```
  else {
    $UNITS[1] = $sn*$UNITS[1] if ($UNITS[1] ne 0);
    $UNITS[2] = $sn*$UNITS[2] if ($UNITS[2] ne 0 && $UNITS[1] eq 0)}
  return @UNITS;
} ##############################################################################

# ============================= CORRECTION IMAGES ==============================#

#--------------------------- Flat Fields MAIN---------------------------------#
sub flat_fields_main { ########################################################
#Main routine to take correction images.
#This is implemented if second line in observing log reads FLAT
#Prompts user to turn on/off light boxes and waits until
#<Enter> is pressed to continue.
#Normal observing continues when finished
    #&slew #-> SLEW TO CORRECT POSITION
&tasks_done("*** Taking correction Images   ***\n");
print "  Halpha Box\n\nTurn on the Halpha light box.  When done press <Enter>.\n";
$input = <STDIN>;  # Waits for input from user and continues when recieved
&Halpha_box; # Halpha images
print "\n\n   Halpha Flats\n\nTurn off the Halpha light box.\nTurn on White box.
When done press <Enter>.\n";
$input = <STDIN>;  # Waits for input from user and continues when recieved
&Halpha_flats; # Halpha flats
print "\n\n   Red-Continuum Flats\n\n";
&r_flats; # Red-continuum Flats
print "\n\n   SII Flats\n\n.";
&sII_flats; # SII flats
print"\n\n\nTurn off all light boxes. Press <Enter> when done to start normal
observing.";
$input = <STDIN>;  # Waits for input from user and continues when recieved
} ##############################################################################
#------ End flat_fields_main --------#

#---------------------------- Halpha Box -------------------------------#
sub Halpha_box { ##############################################################
# Takes 10 Halpha images of the Halpha box.   K.P. Portock 5/17/02
# Camera must be pointed at Halpha box with box turned on.
# Filter must be put on number 2.
# same as the macro HAHA10.CMD.
# Exposure time = 100 milliseconds -> From: NOMINAL SETUP AND OBSERVING PROCEDURE
# Make sure you have run start.cmd prior
# to running this macro.
# K.P. Portock 5/16/02

    &manual_slew(westbox);    # Eastbox chosen arbitrarily
    $exposure = 0.005;    # Exposure time in minutes; 300 ms.
   # Time used during previous observations according to
observing logs
    $image_type = '"other"';    # Define values for fits header info
    $field = '"hbox"';    # Define values for fits header info
    $filter_goal = 2;
    $name = 'hh';    # Info for filename convention
    $flatmessage = 'Halpha light box exposures';
    &expose10;    # exposes 10 images
} ##############################################################################
#-------- End Halpha_box ---------#

#---------------------------- Halpha Flats -------------------------------#
sub Halpha_flats { ############################################################
# Takes 10 Halpha images of the white box (i.e., flats).
# Same as HAFLAT10.CMD   K.P. Portock 5/17/02
# Use filter #2
# Exposure time = 5 seconds -> From: NOMINAL SETUP AND OBSERVING PROCEDURE
```

```
    &manual_slew(eastbox);     # Eastbox chosen arbitrarily
    $exposure = 1.0/12.0;     # Exposure time in minutes; 5.0 s.
    # Time used during previous observations according to
observing logs
    $image_type = '"flat"';     # Define values for fits header info
    $field = '"flat"';     # Define values for fits header info
    $filter_goal = 2;
    $name = 'fh';     # Info for file naming convention
    $flatmessage = 'Halpha flats';
    &expose10;     # exposes 10 images

} ##############################################################################
#-------- End Halpha_flats ----------#

#-------------------------- Red-Continuum flats --------------------------#
sub r_flats { ##################################################################
# Takes 10 red-continuum images of white box (flats)
# Same procedure as RFLAT10.CMD K.P. Portock 5/17/02
# Use filter #1
# Exposure time = 2 seconds -> From: NOMINAL SETUP AND OBSERVING PROCEDURE

    &manual_slew(eastbox);     # Eastbox chosen arbitrarily
    $exposure = 1.0/30.0;     # Exposure time in minutes; 2.0 s.
    # Time used during previous observations according to
observing logs
    $image_type = '"flat"';     # Define values for fits header info
    $field = '"white_box"';     # Define values for fits header info
    $filter_goal = 1;
    $name = 'fr';     # Info for file naming convention
    $flatmessage = 'Red-Continuum flats';
    &expose10;     # exposes 10 images
} ##############################################################################
#----------- end r_flats --------------------#

#---------------------------- SII Flats -----------------------------#
sub sII_flats { ################################################################
# Takes 10 red-continuum images of white box (flats)
# Same procedure as SFLAT10.CMD K.P. Portock 5/17/02
# Use filter #4
# Exposure time = 1 seconds -> chosen for testing purposes
   &manual_slew(eastbox);    # Eastbox chosen arbitrarily
    $exposure = 1.0/30.0;     # Exposure time in minutes; 2.0 s.
    # Time used during previous observations according to
observing logs
    $image_type = '"flat"';     # Define values for fits header info
    $field = '"white_box"';     # Define values for fits header info
    $filter_goal = 4;
    $name = 'fs';     # Info for file naming convention
    $flatmessage = 'SII flats';
    &expose10;     # exposes 10 images
} ##############################################################################
#----------- End sII_flats ------------#

#----------------------- Expose 10 Images --------------------------#
sub expose10 { #################################################################
# Exposes 10 correction images and calls &save_image
# Images are given names = $file_name$n where $pic_num is the number
# of the image in the sequence 1-10. K.P. Portock 5/17/02

&filter($filter_goal);  # Rotate filter wheel and focus
 ###### **** CHECK *** IS FOCUS SAME FOR TAKING THESE IMAGES???
######
$exposure_time = 60000.0*$exposure;  # Exposure time in milliseconds
```

```
$wait_time = $exposure_time/1000+ 20;  # Exposure time in seconds +20 seconds
$hours = (($exposure_time/1000)+ 20)/3600;
@time_lapse = &decimal_converter($hours);# Change format for display purposes
$pic_num = 0;

&tasks_done("* * * $flatmessage * * *");
while(++$pic_num <= 10)  # loops 10 times to take 10
images<---------------modified to 3 for testing
{
  print "\nCapturing image ";  # Status message
  print "(filter #$filter; $exposure min. exposure).\n"; # to screen
  substr($pic_num,0,0) = '0' x (2 - length($pic_num)); # $pic_num = 2-digits
  $file_name = $name.$pic_num.'.fit';  # with leading 0's
  $Client->Execute("OBS $exposure_time");# Tell PMIS to expose for $exposure_time
millliseconds
  # takes a picture, exposure for $exposure milliseconds
  # die "DDE execute failed"; is left out leaving this line in
the code
  # causes an error: "DDE execute failed at PMIS_COM_test.pl
line21."
  printf "Waiting %02d min %02d sec for exposure & image transfer...",
$time_lapse[1], $time_lapse[2];
  # displays PERLs wait time for exposure processing
  sleep($wait_time);   # waits while camera exposes and transfers image
  &save_image;   # calls routine to save the image
  &tasks_done("$flatmessage image saved as $file_name.");  print "\n";
 } # end while( ++$pic_num <= 3 )
tasks_done("$flatmessage are done.\n");
} ##############################################################################
#--------------- End expose10 ------------------#
```

Appendix B

The Polaris Monitor Program

This is the source code for the program which controls the camera that is imaging Polaris. The Polaris monitoring system is to determine when it is cloudy.

```
# Perl script polmonitor_v5.pl
# KPN
# Last modified MONDAY OCTOBER 28, 2002
#
# This is a modified version of the polmonitor.pl program created by
# JHS. This version is designed to be run without user input/control
# (i.e., it does not give a choice as to whether to regulate the CCD
# temperature at ambient or at 15 deg. C below ambient; temp. is reg-
# ulated at ambient - 15C.)
#
# This new version reads the thermoelectric value from the camera and
# converts this to a percentage of power used. It then checks this value
# and adjusts the temperature as necessary to keep the camera operating
# within 50 - 75% of it's maximum power output.
#
# MODIFIED set_temperature subroutine so that it accepts an arguement,
# $delta_temp. Can now be used to regulate the temp to 15 below ambient
# at start and to regulate temperature at specified increments throughout.
#
# Talks to SBIG ST-5 camera.
# Purpose: monitoring the brightness of Polaris.
#
# Usage: At DOS prompt enter...
# perl polmonitor_v5.pl
#
#------ programming comments:
# Perl script to talk to SBIG ST-5 camera.
#
# Instead of waiting after time intensive commands using sleep,
# checks status of command (uses get_activity_status command).
#
# More efficient routine.
# Subroutine(s) used.
# Uses make_and_send_command subroutine.
# Uses a 41 x 41 box around peak in get_minmax
#
# Sends reset command (may be unnecessary).
# Sends take_image command.
# Sends get_readout_peak command (where is Polaris?).
# Sends get_minmax command (for entire image; want mininum).
# Sends get_result_buf command (to obtain get_minmax results).
#


use Win32::SerialPort qw( :STAT 0.19 );

# Any variables I need:

my $file = "st5";
my $cfgfile = $file."_test.cfg";


print "\n\t\t! ! ! POLARIS MONITORING PROGRAM ! ! !\n\n";
print "\nThis program monitors the intensity of the North Star.  This data is";
print "\nused to determine if the skies are clear enough for observations.\n";

# ********* Begin @ 16:30 EST every day **************************
print "\n\n* * * Polaris monitoring begins at 16:30 EST. * * * \n";
#while ($time < 16.50) {&get_time_values}
# ****************************************************************


# Setup:
```

```
# 1) Constructor

$st5port = new Win32::SerialPort ("COM4")
  || die "Can't open $PortName: $^E\n";

# 2) Settings

$st5port->baudrate(9600) || die "bad baudrate";
$st5port->parity('none') || die "bad parity";
$st5port->databits(8) || die "bad databits";
$st5port->stopbits(1) || die "bad stopbits";
# $st5port->buffers(1024,1024);
# default timeouts
$st5port->read_char_time(0);
$st5port->read_const_time(5000);
$st5port->read_interval(0);
$st5port->write_char_time(0);
$st5port->write_const_time(3000);

# 3) Write settings

$st5port->write_settings || undef $st5port;
unless ($st5port) { die "couldn't write_settings"; }

# 4) Save settings in a file

$st5port->save($cfgfile);

# Done with setup.

# Set the bytes for sending the take_image command to the camera:
$b[0] = 0xA5; # Start byte
$b[1] = 0x01; # take_image command byte
$b[2] = 0x1C; # low byte of data length (length = 28)
$b[3] = 0x00; # high byte of data length (length = 28)
#$b[4] = 0x32; # data byte 1: exposure time (low byte), 50 hundredths of a second
$b[4] = 0x2C; # data byte 1: exposure time (low byte), 300 hundredths of a second (1 second)
$b[5] = 0x01; # next exposure time byte (it's a 4 byte "long")
$b[6] = 0x00;
$b[7] = 0x00;
$b[8] = 0x00; # top line to read out (2 byte "int" lowest byte first), 0
$b[9] = 0x00;
$b[10] = 0xF0; # number of lines to read out (2 byte "int"), 240 (ST-5)
$b[11] = 0x00;
$b[12] = 0x00; # leftmost pixel to read out (2 byte "int"), 0
$b[13] = 0x00;
$b[14] = 0x40; # number of pixels to read out (2 byte "int"), 320 (ST-5)
$b[15] = 0x01;
$b[16] = 0x00; # boolean (2 bytes) set to false (0)
$b[17] = 0x00;
$b[18] = 0x00; # boolean (2 bytes) set to false (0)
$b[19] = 0x00;
$b[20] = 0x00; # antiblooming gate state (2 byte "int")
$b[21] = 0x00;
$b[22] = 0xFF; # antiblooming period (2 byte "int")
$b[23] = 0xFF;
$b[24] = 0x01; # destination buffer (2 byte "int") (1 = light frame buffer)
$b[25] = 0x00;
$b[26] = 0x00; # subtract dark? (2 byte boolean)
$b[27] = 0x00;
$b[28] = 0x00; # HIGH readout mode = 0 (2 byte "int")
$b[29] = 0x00;
$b[30] = 0x00; # an ST-6 boolean, set to FALSE here
```

```
$b[31] = 0x00;

# Some constants for conversion of the thermistor reading to Celsius
$t0 = 25.0;
$r0 = 3.0;
$dt = 50.0;
$r_ratio = 9.1;
$r_bridge = 9.09;
$max_ad = 8192.0;

#
# Start controlling the camera.
#

# print "\nSending reset command (1BH) to camera.\n";
$number_of_bytes = 4; # number of bytes in command (excluding checksum bytes).
$b1[0] = 0xA5; # Start byte
$b1[1] = 0x1B; # Command byte
$b1[2] = 0x00; # Number of bytes of data sent (low byte) (None)
$b1[3] = 0x00; # Number of bytes of data sent (high byte)
# &make_and_send_command;
&make_and_send_command(@b1);

system "cls"; #DOS clear screen command
#&get_cpu_info;

# Query user: start regulation of CCD temperature at ambient - 15 degrees (C)?
print "\n";
#print "Regulate CCD temperature at current - 15C ? (Enter 1 or 0 for yes/no)\n";
#print "(0 => CCD temperature regulated at current T, but drops as ambient drops,\n";
#print " 1 => regulated at current T - 15C, *probably* easily accomplished,\n";
#print " reduces noise, generates more heat [enhanced dew prevention?])\n";
#$doit = <STDIN>;
$doit = 1;
if ($doit == 1) {
  &set_temperature(-15);   #Tells camera to drop temperature by -15 C
}
#print "\n";


# Set up the output filename for Polaris monitoring.
&get_time_values;
&create;        # Create observing directory
$date = $year_2."_".$truemonth."_".$mday;
$polmonitor_file = "C:\\SLIC\\DATA\\".$date_name."\\".$date.".POL";
print "Filename: $polmonitor_file\n";
print scalar(localtime), "\n";
$date_today =localtime;
open(POL_FILE, ">$polmonitor_file") || die "Error opening output file. $!\n";
#print POL_FILE "Filename: $polmonitor_file\n";
#print POL_FILE scalar(localtime), "\n";
#if ($doit == 1) {
#  printf POL_FILE "\nThe current CCD temperature (C) is: %.2f\n", $temperature;
#  printf POL_FILE "CCD T regulated to ambient - 15C: %.2f\n", $newtemperature;
#}
if ($doit == 0) {
  print POL_FILE "\nCCD T regulated to current value\n";
  $newtemperature = $temperature;
}
print "\n  i  YYYY MM DD  HH MM SS ZONE  T (C) TE(%) X_pk Y_pk peak  amp     avg\n";
write (POL_FILE_TOP);
close POL_FILE;

# Set up array for running average of amplitude values
```

```
@values=(0,0,0,0,0);  # 5 element array, 5 value running average
$numofvalues = 0;

# Take repeated measurements of the amplitude of Polaris.
$iteration = 0;
while ($iteration < 1000) {   #<------------ TESTING PURPOSES ONLY!!!
#while ($time >= 16.50 || $hour < 8.00) { # Runs from 4:30pm - 8am daily
  $iteration++;
  if($numofvalues < 5) {
    $numofvalues++;
  }
  &get_camera_values;
  &get_time_values;
  $amplitude = $peak - $min;
  $values[4] = $values[3];
  $values[3] = $values[2];
  $values[2] = $values[1];
  $values[1] = $values[0];
  $values[0] = $amplitude;
  $avg = ($values[0]+$values[1]+$values[2]+$values[3]+$values[4])/$numofvalues;
  open(POL_FILE, ">>$polmonitor_file") || die "Error opening output file. $!\n";
  write(POL_FILE);        # Write output to file
  write();        # Write output to screen (STDOUT)
  close POL_FILE;
  # Raise/lower temperature depending on power output. Check every 5th iteration
  # for low-end. Checks every iteration to make sure camera is not overheating.
  &set_temperature(-5) if ($iteration % 5 == 0 && $te_percent < 50);  # Lower temp. in 5-deg
increments
  &set_temperature(5) if ($te_percent > 75);  # Raise temp. in 5-deg C increments

  sleep 60;     #Wait 1 minute
}

# Finally, remove $st5port

undef $st5port;

exec 'perl C:\SLIC\polmonitor_v5.pl'; #Launch program again

# *********


sub get_time_values {
  ($sec,$min,$hour,$mday,$mon,$year,$wday,$yday,$isdst) = localtime(time);
  $trueyear = 1900 + $year;
  $year_2 = $trueyear - 2000;
  $truemonth = 1 + $mon;  # January is $mon = 0
  $time = $hour + $min/60.0;     # decimal hours

# if($var < 10) {$var = "0" . $var}   insures that $var is always 2-digits
  if($truemonth < 10) {$truemonth = "0" . $truemonth}
  if($mday < 10){$mday = "0" . $mday}
  if($year_2 < 10){$year_2 = "0" . $year_2}
  if($min < 10){$min = "0" . $min}
  if($hour < 10){$hour = "0" . $hour}
  if($sec < 10){$sec = "0" . $sec}

  $truewday = 1 + $wday;  # Sunday is $wday = 0
  $trueyday = 1 + $yday;  # Jan 1 is $yday = 0
}

sub get_camera_values {
  #print "\nSending take_image command (01H) to camera.\n";
  $number_of_bytes = 32;
```

```
&make_and_send_command(@b);

# print "\nSending get_activity_status command (05H) to camera (repeatedly).\n";
# ("data" in packet = command to get status of (int), here a 01H)
$packet = chr(0xA5).chr(0x05).chr(0x02).chr(0x00).chr(0x01).chr(0x00).chr(0xAD).chr(0x00);
&check_activity_status;

# print "\nSending get_readout_peak command (03H) to camera.\n";
$packet = chr(0xA5) . chr(0x03) . chr(0x00) . chr(0x00) . chr(0xA8) . chr(0x00);
$count_expected = 12;
&get_result;
$stringfive = substr @strings[0],8,2;
$stringsix = substr @strings[0],10,2;
$stringseven = substr @strings[0],12,2;
$stringeight = substr @strings[0],14,2;
$stringnine = substr @strings[0],16,2;
$stringten = substr @strings[0],18,2;
# print "The peak value, x, y coordinates are:\n";
$peak=hex($stringsix.$stringfive);
# print $peak, "\n";
$peakx=hex($stringeight.$stringseven);
# print $peakx, "\n";
$peaky=hex($stringten.$stringnine);
# print $peaky, "\n";

# print "\nComputing left,top and width,height for use in get_minmax.\n";
# Nominally use a 21 x 21 box centered on peak (max) in image.
#$wide = 21;
#$half = 10;
# Using a 41 x 41 box:
$wide = 41;
$half = 20;
# left:
$value = $peakx - $half;
if ($value < 0) {
  $value = 0;
}
$left = $value;
&int2bytes2dec;
$left1 = $value1;
$left2 = $value2;
# top:
$value = $peaky - $half;
if ($value < 0) {
  $value = 0;
}
$top = $value;
&int2bytes2dec;
$top1 = $value1;
$top2 = $value2;
# width:
$value = $wide; # 320 would be full image, if left value = 0;
if (($left + $wide) > 320) {
  $value = 320 - $left;
}
&int2bytes2dec;
$width1 = $value1;
$width2 = $value2;
# height:
$value = $wide;  # 240 would be full image, if top value = 0;
if (($top + $wide) > 240) {
  $value = 320 - $top;
}
&int2bytes2dec;
```

```
$height1 = $value1;
$height2 = $value2;

# print "\nSending get_minmax command (0CH) to camera.\n";
$number_of_bytes = 14; # number of bytes in command (excluding the 2 checksum bytes)
$b2[0] = 0xA5; # start byte
$b2[1] = 0x0C; # command byte
$b2[2] = 0x0A; # low byte of data length (length = 10);
$b2[3] = 0x00; # high byte of data length (length = 10);
$b2[4] = 0x01; # scan light frame buffer (1)
$b2[5] = 0x00;
#$b2[6] = 0x00; # leftmost pixel, x=0
#$b2[7] = 0x00;
#$b2[8] = 0x00; # top line, y=0
#$b2[9] = 0x00;
#$b2[10] = 0x40; # box width of 320 (full image)
#$b2[11] = 0x01;
#$b2[12] = 0xF0; # box height of 240 (full image)
#$b2[13] = 0x00;
$b2[6] = $left1;
$b2[7] = $left2;
$b2[8] = $top1;
$b2[9] = $top2;
$b2[10] = $width1;
$b2[11] = $width2;
$b2[12] = $height1;
$b2[13] = $height2;
&make_and_send_command(@b2);

# print "\nSending get_activity_status command (05H) to camera (repeatedly).\n";
# (data = command to get status of (int), here a 0CH)
$packet = chr(0xA5).chr(0x05).chr(0x02).chr(0x00).chr(0x0C).chr(0x00).chr(0xB8).chr(0x00);
&check_activity_status;

# print "\nSending command to read result of get_minmax from buffer (15H):\n";
$packet = chr(0xA5) . chr(0x15) . chr(0x00) . chr(0x00) . chr(0xBA) . chr(0x00);
$count_expected = 24;
&get_result;
$stringseven = substr @strings[0],12,2;
$stringeight = substr @strings[0],14,2;
$stringnine = substr @strings[0],16,2;
$stringten = substr @strings[0],18,2;
$stringeleven = substr @strings[0],20,2;
$stringtwelve = substr @strings[0],22,2;
$stringthirteen = substr @strings[0],24,2;
$stringfourteen = substr @strings[0],26,2;
# print "\n";
# print "The min value, x, y coordinates are:\n";
$min=hex($stringfourteen.$stringthirteen.$stringtwelve.$stringeleven);
# print $min, "\n";
$minx=hex($stringeight.$stringseven);
# print $minx, "\n";
$miny=hex($stringten.$stringnine);
# print $miny, "\n";

# print "\nSending read_thermistor command (1DH) to camera.\n";
$packet = chr(0xA5) . chr(0x1D) . chr(0x00) . chr(0x00) . chr(0xC2) . chr(0x00);
$count_expected = 8;
&get_result;
$stringfive = substr @strings[0],8,2;
$stringsix = substr @strings[0],10,2;
# print "The thermistor reading is:\n";
$thermistor=hex($stringsix.$stringfive);
# Convert thermistor reading to Celsius:
```

```
  $r = $r_bridge / (($max_ad/$thermistor) - 1.0);
  $temperature = $t0 - $dt * ( log($r/$r0) / log($r_ratio) );
  &get_temp_status;   # Gets numbers used to compute maximum power usage
}

# ***************************

sub send_command {
  $gotit = "00"; # will stop resending command when receive ACK (0x06)
  while ($gotit ne "06") {
    $st5port->write($packet); # write string to port (i.e., camera)
    select(undef, undef, undef, 0.500);  # wait for 500 milliseconds
    $count_expected = 1; # expecting 1 character in response to reset command
    ($count_in, $string_in) = $st5port->read($count_expected); # read the buffer
    @strings=unpack("H*",$string_in); # Unpack $string_in in hex format, perhaps many bytes
    $gotit = substr @strings[0],0,2;  # cut off first 2 characters (representing first byte)
    # print "Received: ", $gotit, "\n";
  }
}

sub make_and_send_command {
  $sum = 0;
  for ($i=0; $i<$number_of_bytes; $i++) {
    # $sum = $sum + $b[$i];
    $sum = $sum + $_[$i];
  }
  if ($sum > 65536) {
    $sum = $sum - 65536;
  }
  $result = pack("s", $sum);
  #  print "The result of summing the bytes is: ", $result, "\n";
  #  @strings=unpack("H*",$result);
  #  print "The hex unpacked result is: ", @strings, "\n";
  $packet = chr($b[0]);
  for ($i=1; $i<$number_of_bytes; $i++) {
    # $packet = $packet . chr($b[$i]);
    $packet = $packet . chr($_[$i]);
  }
  $packet = $packet . $result;
  @strings=unpack("H*",$packet);
  #  print "The hex unpacked packet is: ", @strings, "\n";
  $gotit = "00"; # will stop resending command when receive ACK (0x06)
  while ($gotit ne "06") {
    $st5port->write($packet); # write string to port (i.e., camera)
    select(undef, undef, undef, 0.500);  # wait for 500 milliseconds
    $count_expected = 1; # expecting 1 character in response to reset command
    ($count_in, $string_in) = $st5port->read($count_expected); # read the buffer
    @strings=unpack("H*",$string_in); # Unpack $string_in in hex format, perhaps many bytes
    $gotit = substr @strings[0],0,2;  # cut off first 2 characters (representing first byte)
    # print "Received: ", $gotit, "\n";
  }
}

sub check_activity_status {
  $status = -1; # will stop resending when receive idle status (=0, int)
  while ($status != 0) {
    select(undef, undef, undef, 0.500);  # wait for 500 milliseconds
    $st5port->write($packet);
    $count_expected = 10;
    ($count_in, $string_in) = $st5port->read($count_expected);
    @strings=unpack("H*",$string_in);
    $string7 = substr @strings[0],12,2;
    $string8 = substr @strings[0],14,2;
    $status=hex($string8.$string7);
```

```
    # print " Status = ", $status, "\n";
  }
}

sub get_result{
  $gotit = "00";
  while (($gotit ne "A5") && ($gotit ne "a5")) {
    $st5port->write($packet);
    select(undef, undef, undef, 0.500);  # wait for 500 milliseconds
    # you must set $count_expected before calling this sub
    ($count_in, $string_in) = $st5port->read($count_expected);
    @strings=unpack("H*",$string_in);
    #  print "The hex unpacked string is: ", @strings, "\n";
    $gotit = substr @strings[0],0,2;
    #  print "Received: ", $gotit, "\n";
  }
}

sub int2bytes2dec {
  # Takes an integer and converts it to two separate bytes (expect <= 65536),
  # each represented by the decimal value equivalent to the byte.
  $res=pack("s", $value);
  @strings=unpack("H*",$res);
  # print "The hex unpacked packet is: ", @strings, "\n";
  $string1 = substr @strings[0],0,2;
  $string2 = substr @strings[0],2,2;
  # print "The substrings (low, high) are: ", $string2, " and ", $string1, "\n";
  $returnval = hex($string2.$string1);
  # print "The returned value = ", $returnval, "\n";
  $value1 = hex($string1);
  $value2 = hex($string2);
}

sub set_temperature {
  my ($delta_temp) = @_;
  # print "\nSending read_thermistor command (1DH) to camera.\n";
  $packet = chr(0xA5) . chr(0x1D) . chr(0x00) . chr(0x00) . chr(0xC2) . chr(0x00);
  $count_expected = 8;
  &get_result;
  $stringfive = substr @strings[0],8,2;
  $stringsix = substr @strings[0],10,2;
  $thermistor=hex($stringsix.$stringfive);
  # print "The thermistor reading is: $thermistor\n";
  # Convert thermistor reading to Celsius:
  $r = $r_bridge / (($max_ad/$thermistor) - 1.0);
  $temperature = $t0 - $dt * ( log($r/$r0) / log($r_ratio) );
  printf "\nThe current CCD temperature (C) is: %.2f\n", $temperature;

  # print "\nSending regulate_temp command (0EH) to camera.\n";
  $number_of_bytes = 16; # number of bytes in command (excluding the 2 checksum bytes)
  $b3[0] = 0xA5; # start byte
  $b3[1] = 0x0E; # command byte
  $b3[2] = 0x0C; # low byte of data length (length = 12);
  $b3[3] = 0x00; # high byte of data length (length = 12);
  $b3[4] = 0x01; # enable temperature regulation (TRUE, 1)
  $b3[5] = 0x00;
  $newtemperature = $temperature + $delta_temp;
   # Setpoint will be 15 degrees below starting ambient.  This value may be
   # adjusted as necessry to keep the camera opeerating within 50 - 75% of
   # its maximum power output.
  if ($delta_temp < 0) {
     printf "The CCD temperature will be dropped to: %.2f\n\n", $newtemperature}
  else {
     printf "The CCD temperature will be raised to: %.2f\n\n", $newtemperature}
```

```
  # Convert Celsius to thermistor value:
  $r = $r0 * exp( log($r_ratio) * ($t0 - $newtemperature) / $dt );
  $realvalue = $max_ad / ( ($r_bridge/$r) + 1.0 );
  $value = int($realvalue);
  # print "Thus the thermistor value will be: $value\n";
  &int2bytes2dec;
  $temp1 = $value1;
  $temp2 = $value2;
  $b3[6] = $temp1;
  $b3[7] = $temp2;
  # print "temp1 and temp2 are $temp1 $temp2\n";
  $b3[8] = 0x0A; # sampling rate, set to 10 as recommended in ST-5 manual
  $b3[9] = 0x00;
  $b3[10] = 0xE8; # p_gain, set to 1000 as recommended
  $b3[11] = 0x03;
  $b3[12] = 0xA4; # i_gain, set to 164 as recommended
  $b3[13] = 0x00;
  #$b3[10] = 0xA4; # p_gain, set to 164 as recommended
  #$b3[11] = 0x00;
  #$b3[12] = 0xE8; # i_gain, set to 1000 as recommended
  #$b3[13] = 0x03;
  $b3[14] = 0x01; # reset_brownout set to TRUE (1)
  $b3[15] = 0x00;
  &make_and_send_command(@b3);
}

sub get_cpu_info {
#Send the get_cpu_info command (25H) to get various pieces of information
#about the camera/cpu settings
  #    print "\nSending get_cpu_info command (25H) to camera.\n";
  #    print "\nSending read_thermistor command (1DH) to camera.\n";
  $packet = chr(0xA5) . chr(0x25) . chr(0x00) . chr(0x00) . chr(0xCA) . chr(0x00);
  $count_expected = 62;   # 62 = 56 data bytes + 6 standard packet bytes
  &get_result;
  $stringfive = substr @strings[0],8,2;
  $stringsix = substr @strings[0],10,2;
  $stringseven = substr @strings[0],12,2;
  $stringeight = substr @strings[0],14,2;
  $version=hex($stringsix.$stringfive);
  $cpu=hex($stringeight.$stringseven);
  # print "The version reading is: $version\n";
  # print "The cpu is: $cpu\n";
  $stringnine = substr @strings[0],104,2;
  $stringten = substr @strings[0],106,2;
  $max_te=hex($stringten.$stringnine);
  # print "The max_te_drive reading is: $max_te\n";
  $stringeleven = substr @strings[0],108,2;
  $stringtwelve = substr @strings[0],110,2;
 $image_width=hex($stringtwelve.$stringeleven);
 #     print "The image_width reading is: $image_width\n";
}

sub get_temp_status {
  #Send the get_temp_status command (20H)
  $packet = chr(0xA5) . chr(0x20) . chr(0x00) . chr(0x00) . chr(0xC5) . chr(0x00);
  $count_expected = 20;   # 20 = 14 data bytes + 6 standard packet bytes
  &get_result;
  $stringseven = substr @strings[0],16,2;
  $stringeight = substr @strings[0],18,2;
  $te_value=hex($stringeight.$stringseven);
  # print "The te_value reading is: $te_value\n";
  $te_percent = $te_value/19.0;
  # print "The te_percent reading is: $te_percent\n";
```

```
  $stringnine = substr @strings[0],20,2;
  $stringten = substr @strings[0],22,2;
  $sample_rate=hex($stringten.$stringnine);
  # print "The sample_rate is: $sample_rate\n";
}

sub create {
#Creates a unique label - using the format YY_MM_DD.  It then uses this label to
#create and name the directory where the evening's observations will be stored. It
#also creates the log file - YY_MM_DD.LOG - for the evening's observing session.
#
#Create the name label
  @date = localtime; #Reads local time from system clock
  $year = $date[5] - 100.0; #2-digit year
  $year4 = $year + 2000.0; #Some calculations require a 4-digit year
  $month = $date[4] + 1.0; #month
  $day = $date[3]; #day of month
  substr($month,0,0) = '0' x (2-length($month)); #Gives 2-digit month
  substr($day,0,0) = '0' x (2-length($day));  #Gives 2-digit day
  substr($year,0,0) = '0' x (2-length($year));  #Gives 2-digit year
  $date_name = $year.'_'.$month.'_'.$day;  #Directory/log file name

#Create the directory
  $directory = 'C:\SLIC\DATA\\'.$date_name;     #name of directory: C:\SLIC\YY_MM_DD
  mkdir($directory);  #create evening's directory if it doesn't exist
  chdir($directory);  #makes this the current directory

#Create log file named YY_MM_DD.LOG
  $log_file = $date_name.'.LOG';
  open(LOGFILE,">$log_file") || die("Cannot Open File!");
  print (LOGFILE "Polaris monitor started ", scalar(localtime), " EST.\n\n");
  close (LOGFILE);  #Close log_file
}

# Format statements for formatting output data in the file and on the screen
format POL_FILE =
@>>>  @>>> @> @>  @> @> @> @||  @##.## @##.##  @>>   @>>   @####    @>>>>   @####.##
$iteration $trueyear $truemonth $mday $hour $min $sec "EST" $temperature $te_percent $peakx
$peaky $peak $amplitude $avg
.

format STDOUT =
@>>> @>>> @> @>  @> @> @> @|| @##.## @##.##  @>> @>>  @####  @>>>>  @####.##
$iteration $trueyear $truemonth $mday $hour $min $sec "EST" $temperature $te_percent $peakx
$peaky $peak $amplitude $avg
.

format POL_FILE_TOP =
Filename: @>>>>>>>>>>>>>>>>>>>>>
  $polmonitor_file
Today's Date: @>>>>>>>>>>>>>>>>>>>>>>
      $date_today

The current CCD temperature is: @##.## C
$temperature
CCD temperature regulated to: @##.## C
      $newtemperature

  i   YYYY MM DD  HH MM SS ZONE   T (C)  TE(%)  X_pk  Y_pk   peak     amp   avg
====  ==== == ==  == == == ====   =====  =====  ====  ====   ====    =====   ========
.
```

Appendix C

The Mount Alignment Program

What follows is the source code for a program which aids in alignment of the mount. This program allows the user to command the mount to point using RA/DEC or ALT/AZ coordinate systems. It also allows the user to slew the mount in specific directions such as north, south, east and west.

```
#!/usr/bin/perl
# mount_park.pl       K. P. Portock       Last Modified 8/27/02
# Testing PERL script with losmandy mount.
# this Progam communicates and controls the Losmandy Gemini
# mount by sending LX200 commands.  It also receives data
# from the mount after certain commands.  See Gemini Level2 command set
# for details.
# This program stops the Gemini mounts from tracking,
# then waits, and restarts tracking.
# Test for telescope parking.

use Win32::SerialPort qw( :STAT 0.19 );

$mount_port = new Win32::SerialPort ("COM1") || die "Can't Open $PortName: $^E\n";
      #serial port name is $mount_port

#  Configure Serial Port

$mount_port->baudrate(9600)         || die "bad baudrate";
$mount_port->parity('none')         || die "bad parity";
$mount_port->databits(8)            || die "bad databits";
$mount_port->stopbits(1)            || die "bad stopbits";
#$mount_port->buffers(1024,1024);
# default timeouts
$mount_port->read_char_time(0);
$mount_port->read_const_time(5000);   # Read command waits 5 seconds.
$mount_port->read_interval(0);
$mount_port->write_char_time(0);
$mount_port->write_const_time(3000);

$mount_port->write_settings         || undef $mount_port;
unless ($mount_port) {die "couldn't write_settings";}

# ------------------------------------------------------ #

   $slew_error = 0;                 # Initialize variable
   $slew_error2 = 0;

&initialize_mount;
&get_command;
&mount(park);
&shut_down;

# ------------------------------------------------------ #
# --- Check RA & DEC--- #
sub check_RA_DEC
{
   $tolerance = 5;
   print "\nChecking if coordinates are correct...   slew_error = $slew_error\n";
   if ( ($current_RAh == $RAh) && ($current_DECd == $DECd)  )       # RA hour & DEC degrees must
be exact
      { if ( ($current_RAm == $RAm) && ($current_DECm == $DECm) )   # RA & DEC minutes must be
exact
           {
           #            if ( (abs($current_RAs - $RAs) <= $tolerance)            # error in RA
seconds,
           #                  &&  (abs($current_DECs - $DECs) <= $tolerance) )   # & error in
DEC seconds must be less than some tolerance
           #                {
            $slew_error = 0;                 # reset error counter
            print "\nCoordinates are correct.\n";
            print "\n\n";
            print "\nRA  $current_RAh:$current_RAm:$current_RAs    DEC
$current_DECd:$current_DECm:$current_DECs\n";
```

```
            sleep(2);
            # &tasks_done("Slew complete");
            # &tasks_done("RA  $current_RAh:$current_RAm:$current_RAs    DEC
$current_DECd:$current_DECm:$current_DECs");
            # &tasks_done("Slew complete");
            return;
           #                 } # end if ( (abs($current_RAs - $RAs)<= $tolerance) &&
(abs($current_DECs - $DECs)  <= $tolerance) )
           #            else { ++$slew_error; }               # If seconds not correct try slew
again
            } # end  if ( $current_RAm == $RAm )
        else {  ++$slew_error;            # If minutes not correct try slew again
                print "\nslew_error != 0 HOO HAA     slew_error = $slew_error";
                if ($slew_error <= 2 )
                   { print "\n slew_error <= 2 HOO HAA    slew_error = $slew_error\n";
$slew_error2 = 1; return;}
                else { $mount_error = 'error';
                       print "\n*** MOUNT MALFUNCTION ***\n";
                       print "COULD NOT SLEW TO $RAh:$RAm:RAs RA, $DECd:$DECm:DECs DEC.\n";
                       # &tasks_done ("*** MOUNT MALFUNCTION ***");
                       # &tasks_done ("COULD NOT SLEW TO $RAh:$RAm:RAs, $DECd:$DECm:DECs.");
                       &shut_down;
                      } # end else
             } # end else
       } # end if ($current_RAh == $RAh)
   else { ++$slew_error;                   # If hour not correct try slew again
          print "\nslew_error != 0 HOO HAA     slew_error = $slew_error";
          if ($slew_error <= 2 )
             { print "\n slew_error <= 2 HOO HAA    slew_error = $slew_error\n"; $slew_error2 =
1; return;}
          else { $mount_error = 'error';
                 print "\n*** MOUNT MALFUNCTION ***\n";
                 print "COULD NOT SLEW TO $RAh:$RAm:RAs RA, $DECd:$DECm:DECs DEC.\n";
                 # &tasks_done ("*** MOUNT MALFUNCTION ***");
                 # &tasks_done ("COULD NOT SLEW TO $RAh:$RAm:RAs, $DECd:$DECm:DECs.");
                 &shut_down;
                } # end else
         } # end else

} # end sub check_RA_DEC

# -------------------- Get command from user --------------------- #
sub get_command
 {
    Test_top:
    print "\n M - Move telescope \n P - Park \n Q - Quit Moving \n S - Slew To Coordinates \n T
- Track \n X - Exit program\n";
    print "\n Enter Command:";                        #get command to move or stop
    $inst = <>;
    chomp $inst;
     if ($inst eq 'Q' || $inst eq 'q')               # Quit movements command received
        {  $response_count = 0;                      # Number of characters in Geminis reply.
Reply depends on command sent.
           $mount_string = 'Telescope has stopped';  # Code is Gemini's cryptic reply to this
command. 131s-> sidereal 135w->no tracking
           $mount_command = ':Q#';                   # Form proper command to request tracking
speed
           &send_to_mount;
         } # end if ($input eq 'Q' || $dir eq 'q')
     elsif ($inst eq 'M' || $inst eq 'm')            # Move command received
        {  Move_top:
           #$dir = '';
           print "\n N - Move North \n E - Move East \n S - Move South \n W - Move West \n";
           print " Enter direction to move:";
```

```
          $dir = <>;
          chomp $dir;
           if ($dir eq 'N' || $dir eq 'n')           # Move North command received
           {  $response_count = 0;
              $mount_command = ':Q#';                # Stop previous movements before starting
this one
              &send_to_mount;
              $mount_string = 'Moving North';        # Move North
              $mount_command = ':Mn#';
              &send_to_mount;
            } # end if ($dir eq 'N'  || $dir eq 'n')
           elsif($dir eq 'E' || $dir eq 'e')         # Move East command received
           {  $response_count = 0;
              $mount_command = ':Q#';                # Stop previous movements before starting
this one
              &send_to_mount;
              $mount_string = 'Moving East';         # Move East
              $mount_command = ':Me#';
              &send_to_mount;
            } # end elsif($dir eq 'E'  || $inst eq 'e')
           elsif($dir eq 'S'  || $dir eq 's')        # Move South command received
           {  $response_count = 0;
              $mount_command = ':Q#';                # Stop previous movements before starting
this one
              &send_to_mount;
              $mount_string = 'Moving South';        # Move South
              $mount_command = ':Ms#';
              &send_to_mount;
            } # end elsif($dir eq 'S'  || $dir eq 's')
           elsif($dir eq 'W' || $dir eq 'w')         # Move west command received
           {  $response_count = 0;
              $mount_command = ':Q#';                # Stop previous movements before starting
this one
              &send_to_mount;
              $mount_string = 'Moving West';         # Move West
              $mount_command = ':Mw#';
              &send_to_mount;
            } #end elsif($dir eq 'W' || $dir eq 'w')
           else                                      # Move direction command not recognized
           {  print "\n 2 Ooops...you typed the wrong letter.\n";
              goto Move_top;
            } #end else
        } #end elsif ($input eq 'M' || $inst eq 'm')
      elsif ($inst eq 'P' || $inst eq 'p')           # Park command received
         {  &mount(park);  }
      elsif ($inst eq 'T' || $inst eq 't')           # Track command received
         {  &mount(track); }
      elsif ($inst eq 'S' || $inst eq 's')           # Slew command recieved
         {  &get_coord;
            SLEW_IT:
             @coordinates = ($RAh, $RAm, $RAs, $DECd, $DECm, $DECs);
             &slew_telescope(@coordinates);   # formats commands to send to gemini Slew
telescope
             &send_and_check_command;         # Send slew command to gemini
             print "\nSlew_error2 = $slew_error2\n\n";
             #if ($slew_error2 = 1){ goto SLEW_IT;}
          } # end elsif ($inst eq 'S' || $inst eq 's')
      elsif ($inst eq 'X' || $inst eq 'x')           # Exit command recieved
         {  exit;  }
      else                                           # Command not recognized
         { print " 1 Ooops....you typed the wrong letter.\n";
           goto Test_top;
          }
    goto Test_top;
```

```
 } # end sub get_command

# ---------------- Get corrdinates from user ---------------------#
sub get_coord
{
    print "\n\nType RA hours (HH) and press return: ";
    $RAh = <>;
    print "Type RA minutes (MM) and press return: ";
    $RAm = <>;
    print "Type RA seconds (SS)and press return : ";
    $RAs = <>;
    print "Type DEC degrees (+/-dd) and press return: ";
    $DECd = <>;
    print "Type DEC minutes (mm) and press return: ";
    $DECm = <>;
    print "Type DEC seconds (ss) and press return: ";
    $DECs = <>;

} # end  sub get_coord

# -------------------------- initialize mount ------------------------------ #
sub initialize_mount    # initializes gemini mount setting.  Time, time zone, lattitude &
longitude.
{
    #--- Get date and time from PC.  Format date and time. ----#
    @date = localtime;                      # Reads local time & date from system clock
    $year = $date[5]-100;                           # 4-digit year
    $month = $date[4]+1;              # Month
    $day = $date[3];                      # Day of month
    substr($month,0,0) = '0' x (2-length($month));   # Gives 2-digit month
    substr($day,0,0) = '0' x (2-length($day));      # Gives 2-digit day
    substr($year,0,0) = '0' x (2-length($year));     # Gives 2-digit year
    $date = "$month/$day/$year";                     # Date in mm/dd/yy format

    $local_time = localtime;
    $gm_time = gmtime; @gm_time = gmtime;
    $time_zone = $gm_time[2]- $date[2];              # Calculates difference in local time and
GMT in hours to get time zone
    $hour = $date[2];                                # Get hour
    $minute = $date[1];                              # Get minute
    $second = $date[0];                              # Get second
    substr($hour,0,0) = '0' x (2-length($hour));     # Gives 2-digit hour
    substr($minute,0,0) = '0' x (2-length($minute)); # Gives 2-digit minute
    substr($second,0,0) = '0' x (2-length($second)); # Gives 2-digit second
    $l_time = "$hour:$minute:$second";               # Local time in hh:mm:ss format

    print "\n\n\nl_time: $l_time";
    print "\nLocal time: $local_time";               # These statements for debuggin'
    print "\ntime : $l_time";
    print "\nGM time: $gm_time";
    print "\nTime zone: $time_zone\n\n";

    $string = 'Time zone set';
    $response_count = 1;                    # Number of characters in Geminis reply.  Reply
depends on command sent.
    $mount_command = ":SG$time_zone#";       # Form proper LX200 command to SET TIME ZONE for
Gemini Mount
    &send_to_mount;                          # Send command

    $mount_string = 'Local time set';
    $response_count = 1;                    # Number of characters in Geminis reply.  Reply
depends on command sent.
    $mount_command = ":SL$l_time#";          # Form proper LX200 command to SET LOCAL TIME for
Gemini Mount
```

```
    &send_to_mount;                          # Send command

    $mount_string = 'Calendar date set';
    $response_count = 50;                    # Number of characters in Geminis reply.  Reply
depends on command sent.
    $mount_command = ":SC$date#";            # Command to SET DATE for Gemini Mount
    &send_to_mount;                          # Send command
    #-------Confirm date data and display on screen--------#
    $mount_string = 'date';
    $response_count = 9;                     # Number of characters in Geminis reply.  Reply
depends on command sent.
    $mount_command = ":GC#";                 # Command to GET CALENDAR DATE from Gemini Mount
    &send_to_mount;                          # Send command

    $mount_string = 'Time';
    $response_count = 9;                     # Number of characters in Geminis reply.  Reply
depends on command sent.
    $mount_command = ":GL#";                 # Command to GET TIME from Gemini Mount Displays in
GMT
    &send_to_mount;                          # Send command
    #----Set observing site longitude and latitude-------#
    $mount_string = 'Observing site longitude set';
    $response_count = 1;                     # Number of characters in Geminis reply
    $mount_command = ":Sg+080*33#";          # Command to SET LONGITUDE of observing site.
Format:  +/-ddd*mm
    &send_to_mount;                          # Long. of SLIC (80d 33m 30s W)

    $mount_string = 'Observing site latitude set';
    $response_count = 1;                     # Number of characters in Geminis reply
    $mount_command = ":St+37*20#";           # Command to SET LATITUDE of observing site.
Format:  +/-dd*mm
    &send_to_mount;                          # Lat. of SLIC (37d 19m 48s N)

} # end sub initialize_mount

#---------------------mount park or track---------------------------------#
sub mount
 # This subroutine parks the mount OR starts tracking:
 #   - If it is sent "park" it will stop mount tracking.
 #     Can slew to any position before of after parking
 #     and mount will not track.  Command ->  &mount(park);
 #   - If it receives "track" it will initiate sidereal
 #     tracking.  MOUNT WILL REMAIN STOPPED, AFTER TRACKING
 #     IS INITIATED, UNTIL A SLEW OR MOVE COMMAND IS SENT.
 #     MUST SLEW AFTER TRACKING IS INIATED TO ACTIVATE TRACKING.
 #     Command -> &mount(track);
   {
     my($action) = @_;     # Get action desired.  $action = "park" to stop tracking.
                           #                      $action = "track" to initiate tracking.
     $error = 0;
     $mount_error = 'none';
# ----- Send and check park commands ----- #
     if ($action eq "park")
        { Park:
          $response_count = 0;                        # No reply to this command.
          $mount_string = 'parking sequence initiateed';
          $mount_command = '>135:s#';                 # Form proper command to stop tracking.
          &send_to_mount;                             # send command to mount.
 #----- Check if park command was obeyed ----- #
          $response_count = 5;                        # Number of characters in Geminis
response.
          $mount_string = 'Tracking Speed Code';      # Code is Gemini's cryptic response to
this command. 131s-> sidereal 135w->no tracking.
          $mount_command = '<130:t#';                 # Form proper command to request
```

```
tracking speed.
           &send_to_mount;                              # send command to mount.
           if ($response ne "135w#")                    # Check if proper command was received.
              {  print "\nPark command not received\n";
                 if ( $error < 3 )                      # Try to park 3 times if response not
acceptable.
                     { ++$error;
                       print "Trying again...";
                       goto Park;                       # Try again.
                      }# end if ( $error < 3 )
                 else   # Failed to park 3 times in a row, so shut down.
                     { $mount_error = 'error';
                       #&tasks_done("*** MOUNT MALFUNCTION! ***");
                       #&tasks_done("Parking failed.");
                       &shut_down;
                      }# end else
               } # end if ($response ne "135w#")
           else  #$response equals "135w#"  Correct parking response received.
                { $error = 0;
                  #&tasks_done("Park successful.");
                  print "\nTelescope is parked.\n";
                 }# end else
         }  #end if ($action eq "park")
# ----- Send and check track commands ----- #
      elsif ($action eq "track")
         { Track:
           $response_count = 0;                         # Number of characters in Geminis
response.  response depends on command sent.
           $mount_string = 'Sidereal tracking mode set';
           $mount_command = '>131:w#';                  # Form proper command to set sidereal
tracking mode.
           &send_to_mount;                              # IMPORTANT: Must Slew to new field
AFTER sidereal tracking.
 #----- Check if track command was obeyed ----- #      # is set!  Mount does not activate
tracking until.
           $response_count = 5;                         # Number of characters in Geminis
response.  response depends on command sent.
           $mount_string = 'Tracking Speed Code';       # Code is Gemini's cryptic response to
this command. 131s-> sidereal 135w->no tracking.
           $mount_command = '<130:t#';                  # Form proper command to request
tracking speed.
           &send_to_mount;                              # Send command to mount.
           if ($response ne "131s#")                    # Check if proper command was received.
              { print "\nTrack command not received\n";
                if ( $error < 3 )           # Try to track 3 times if response not acceptable.
                    { ++$error;
                      print "Trying again...";
                      goto Track;            # Try again.
                     }# end if ( $error < 3)
                else   # Failed to park 3 times in a row, so shut down.
                    { $mount_error = 'error';
                      # &tasks_done("*** Mount Malfunction ***");
                      # &tasks_done("Track initiation failed.");
                      &shut_down;
                     }# end else
               }# end if ($response ne "135w#")
           else #$response equals "131s#"  Correct tracking response received.
                { $error = 0;
                  print "\nTelescope tracking initiated.\n";
                  # &tasks_done("Telescope tracking initiated.");
                 }# end else
         }# end elsif ($action eq "track")
 # ---- Command not recognized ---- #
      else  # $action not equal to "park" AND $action not equal to "track"
```

```
        { print "\nCommand Not Recognized.\n";
          $mount_error = 'error';
          # &tasks_done("*** MOUNT MALFUNCTION ***");
          # &tasks_done("Mount Park or track command not recognized.");
          &shut_down;
         }# end else
    } #end sub mount


 #-------------------------------------------------------------------------------#
sub send_and_check_command
{
  # ----- Send and check command to slew to selected field ----- #
   $response_count = 1;                         # For this command, only get 1st character of
the response to
   $mount_string = 'Sending slew command';                  # determine what the response is in
the 'if' statements that follow.
   $mount_command = ':MS#';                     # Tell gemini Mount to Slew to the selected
object
   &send_to_mount;
   $slew_error2 = 0;
   # This block of 'if' statements is to check the response of the Gemini mount to
   # the slew command ':MS#'.
   if ($response == 0 )                          # ******** Response '0'  if command was
accepted ********
      { print "  Slewing Telescope...\n";
        # &tasks_done("Slewing Telescope...");
        $mount_error = 'none';
        $error = 0;
        print "\n\n Now I become THE SLEW MASTER!!!! \n";   #degubbing
sleep(1); # debugging
        &slew_master;               # watches RA and DEC until two consecutive values are equal
        print "\nSlew_master is done\n";
        &check_RA_DEC;              # checks that the current RA and DEC are the desired RA and
DEC
        return;
       }# end if ($response == 0 )
   elsif ($response == 1 )                       # ******* Response is '1Object below horizon.#'
 *********
      { print"\n NOW I AM HERE HOO HOO HAA HAA!!!\n";
        $response_count = 22;                    # Get remaining response from gemini.
        ($count_in, $response) = $mount_port->read($response_count);
        $mount_error = 'error';                  # ERROR slewing mount: take action!
        print "whoops...$response\n Try another field.\n";  # show response
        # &tasks_done ("whoops...$response\n Try another field.");  # Print response
        # $pic_num = $num_pics;                  # Goes to next field
        return;
       }# elsif ($response == 1 )
   elsif ($response == 2 )                       # Gemini sends two strings that start with 2...
      { $response_count = 1;                     # Get next character to determine response.
        ($count_in, $response) = $mount_port->read($response_count);
        if ($response eq "T")                    # ******** Response is '2Telescope is not
aligned.#' *******
             { $response_count =  23;            # Get remaining response from gemini.
               ($count_in, $response) = $mount_port->read($response_count);
               $mount_error = 'error';           # ERROR slewing mount: take action!
               print "\n***  Ooops...T$response ***\n";
               # &tasks_done("*** MOUNT MALFUNCTION ***");
               # &tasks_done("oops...T$response");
               &shut_down;
              }# end  if ($response eq "T")
        elsif ($response eq "N")                 # ******** Response is '2No object selected.#'
********
             {  $response_count = 19;            # Get remaining response from gemini.
```

```
               ($count_in, $response) = $mount_port->read($response_count);
               $error++;
               if ( ++$error <= 3 )            # Try to slew 3 times if '2No object
selected.#' occurs.
                    { print "\nOoops...N$response.";
                      print "  Trying again...";
                      # &tasks_done("\nOoops...N$response.  Trying again...");
                      $slew_error2 = 1;                # Make perl Try again.
                      return;
                     }# end if ( ++$error <= 3 )
               else
                    { $mount_error = 'error';   # ERROR slewing mount: take action!
                      # &tasks_done("*** MOUNT MALFUNCTION ***");
                      # &tasks_done("*** Uh-oh... N$response ***");
                      print "\n*** MOUNT MALFUNCTION ***";
                      print "\n*** Uh-oh... N$response ***\n";
                      # &tasks_done("\n*** MOUNT MALFUNCTION ***");
                      # &tasks_done("\n*** Uh-oh... N$response ***\n");
                      &shut_down;
                     }# end else
             }# end elsif ($response eq "N")
        else
             {  $mount_error = 'error';
                # &tasks_done("*** MOUNT MALFUNCTION ***");
                # &tasks_done("Slew command not recognized.);
                print "\n*** MOUNT MALFUNCTION ***";
                print "\nSlew command not recognized.\n";
                &shut_down;
             } # end else
       } # end elsif ($response == 2 )
   elsif ($response == 3 )                      # ******* Response is '3Manual control.#'
*********
      {  $response_count = 16;                  # Get remaining response from gemini.
         ($count_in, $response) = $mount_port->read($response_count);
         $mount_error = 'error';                # ERROR slewing mount: take action!
         # &tasks_done("*** MOUNT MALFUNCTION ***");
         # &tasks_done("Error slewing mount. $response.");
         print "*** MOUNT MALFUNCTION ***";
         print "Error slewing mount. $response.";
         &shut_down;
       } # end elsif ($response == 3 )
   else
      {  $mount_error = 'error';
         print "Command not recognized.";
         # &tasks_done("*** MOUNT MALFUNCTION ***");
         # &tasks_done("Slew command not recognized.);
         &shut_down;
       } # end else
  }# end sub send_and_check_command


# ------------------- Send and check DEC command ------------------------- #
sub SEND_DEC
{
   $response_count = 1;                         # Number of characters in Geminis response.
response depends on command sent.
   $mount_string = 'DEC sent';                  # Declination of the field, DEC must be in
+/-dd:mm:ss format
   $mount_command = ":Sd$DECd:$DECm:$DECs#";    # Form proper command to set new DEC
   &send_to_mount;
   if ($response != 1)                          # Check for correct mount response.
      { ++$error;                               # Response of 1 means valid command,
        print "\nError sending DEC command.";   # see gemini level2 command set for details.
        if ($error <= 3)                        # If wrong response the try again
```

```
          { print "  Trying again...";          # Try 3 times then shut down
            &SEND_DEC;
           }# end if ($error <= 3)
        else
          { $mount_error = 'error';            # Messed up three times, shut down.
            # &tasks_done("*** MOUNT MALFUNCTION ***");
            # &tasks_done("*** Error sending DEC command to Gemini mount***");
            print "\n*** MOUNT MALFUNCTION ***";
            print "\n*** Error sending DEC command to Gemini mount***\n";
            &shut_down;
           }# end else
       } # end if ($response != 1)
   else { $error = 0;}
} # end sub SEND_DEC
#-------------------------------------------------------------------------------#
     # ----- Send command to name the selected object ----- #
sub send_name
{
   $response_count = 0;                        # Number of characters in Geminis response.
response depends on command sent.
   $mount_string = 'Name sent';
   $field = 'Obi-Wan Kenobi';                  # Name of selected object, used for testing
purposes
   $mount_command = ":ON$field#";              # Tell Gemini Mount the name of the selected
object
   &send_to_mount;
} # end sub send_name


#-------------------------------------------------------------------------------#
sub SEND_RA  # Label for mount goto statement
{
  # RA & DEC given as: $RAh:$RAm:$RAs, $DECd:$DECm:$DECs
  # ----- Send and check RA command ----- #
   $response_count = 1;                        # Number of characters in Geminis response.
response depends on command sent.
   $string = 'RA sent';                        # Right Ascension of the field, RA Must be in
hh:mm:ss format
   $mount_command = ":Sr$RAh:$RAm:$RAs#";      # Form proper command to set the new RA
   &send_to_mount;
   if ($response != 1)                         # Check for correct mount response.
      { ++$error;                              # Response of 1 means valid command,
        print "\nError sending RA command.";   # see gemini level2 command set for details.
        if ($error <= 3)                       # If wrong response the try again
           { print "  Trying again...";          # Try 3 times then shut down
             &SEND_RA;
            }# end if ($error <= 3)
        else
           { $mount_error = 'error';           # Messed up three times, shut down.
             # &tasks_done("*** MOUNT MALFUNCTION ***");
             # &tasks_done("Error sending RA command to Gemini mount");
             print "\n*** MOUNT MALFUNCTION ***";
             print "\n*** Error sending RA command to Gemini mount ***\n";
             &shut_down;
            }# end else
       }# end if ($response != 1)
   else { $error = 0;}

} # end sub SEND_RA

#---------------------- Send Command to Gemini Mount ----------------------#
# Must set $mount_command to valid form.
# Set $string to desired output.
sub send_to_mount
```

```
    {
     $mount_port->write($mount_command);                              # Send command.
     ($count_in, $response) = $mount_port->read($response_count);   # Read Geminis response, see
Gemini level2 command set
     print "$mount_string: $response\n";                              # For Debuggin'
     #if ( length($response) != $response_count )                     # Check if response from
Gemini has the expected
     #   { $mount_error = 'error';                                    # number of characters.  If
it does not then there
     #     # &tasks_done("*** MOUNT MALFUNCTION ***");                # has been a malfuntion, so
shut down
     #     # &tasks_done("Gemini Mount did respond correctly");
     #     print "\n*** MOUNT MALFUNCTION ***\n";                     # For debugging
     #     print "\nGemini Mount did respond correctly\n";
     #     print "\n$response_count characters expected.\n";          # For debuggin'
     #     print "\nlength($response) characters received.\n";       # debug
     #     &shut_down;
     #   } # end if ( length($response) != $response_count )
    }# end sub send_to_mount
#------------ routine for testing -------------------------#
sub shut_down
{
  print "\n\nmount_error = $mount_error";
  print "\nslew_count = $slew_count";
  print "\nslew_error = $slew_error";
  print "\nerror = $error";
  print "\n*** Shutting Down ***\n";
  exit;
}# end sub shut_down

#----------------------- The Slew Master ----------------#
sub slew_master
# This subroutine montitors the progress of mount as it slews to new coordinates
# It loops and periodically checks the RA and DEC of the mount and compares it with the
# previous check.  If the RA and DEC haven't changed for 3 consecutive checks then it assumes
# the mount is finished slewing.  Then it compares the actual RA and DEC with the RA and DEC
# needed for a particular field.  If they agree within some tolerance then observing can
continue.
# if not then the slew is performed again.
{
# --- Monitor RA & DEC Slew --- #
  $last_RA = 'none';                              # Initialize variables
  $last_DEC = 'none';

  for ($slew_count = 0; $slew_count <= 2; ++$slew_count)  # Loop until 3 consecutive values of
RA are equal
    { sleep(1);                                   # Wait 1 second

      $response_count = 9;                        # Number of characters in Geminis response.
response depends on command sent.
      $mount_string = 'Current_RA';                    # For debuggin'
      $mount_command = ":GR#";                    # Form proper command to get the current RA
      &send_to_mount;                             # Send command to get RA
      $current_RA = $response;                    # Set current_RA equal to response from
Gemini

      $response_count = 10;                       # Number of characters in Geminis response.
Response depends on command sent.
      $mount_string = 'Current_DEC';                   # For debuggin'
      $mount_command = ":GD#";                    # Form proper command to get the current DEC
      &send_to_mount;                             # Send command to mount to get current DEC
      $current_DEC = $response;                   # Set current_DEC equal to response from
Gemini
      print "In FOR loop   slew_count = $slew_count\n\n";
```

```
      while ( ($current_RA ne $last_RA) || ($current_DEC ne $last_DEC) ) # Loop until two
consecutive values of RA &DEC are equal
          { $slew_count = 0;                      # Consecutive values are not equal, reset
$slew_count
            $last_RA = $current_RA;               # Reset $last_RA
            $last_DEC = $current_DEC;             # Reset $last_DEC
            sleep(1);                             # Wait 1 second

            $response_count = 9;                  # Number of characters in Geminis response.
            $mount_string = 'Current_RA';               # For debuggin'
            $mount_command = ":GR#";              # Form proper command to get the current RA
            &send_to_mount;                       # Send command to get RA
            $current_RA = $response;              # Set current_RA equal to response from
Gemini

            $response_count = 10;                 # Number of characters in Geminis response.
            $mount_string = 'Current_DEC';              # For debuggin'
            $mount_command = ":GD#";              # Form proper command to get the current DEC
            &send_to_mount;                       # Send command to mount to get current DEC
            $current_DEC = $response;             # Set current_DEC equal to response from
Gemini
    print "\nin while loop  slew_count = $slew_count\n\n";
          } # end while ( $current_RA ne $last_RA )
     } # end for ($slew_count = 0; $slew_count <= 3; ++$slew_count)

  $current_RAh = substr($current_RA, 0, 2);       # Split & format current_RA hour  HH
  $current_RAm = substr($current_RA, 3, 2);       # Split & format current_RA minutes MM
  $current_RAs = substr($current_RA, 6, 2);       # Split & format current_RA seconds  SS

  $current_DECd = substr($current_DEC, 0, 3);     # Split &format current_DEC degrees  +/-dd
  $current_DECm = substr($current_DEC, -6, 2);    # Split & format current_DEC minutes  mm
  $current_DECs = substr($current_DEC, -3, 2);    # Split & format current_DEC seconds  ss

  print "\n Three consecutive values of RA and DEC are equal.  YIPPEEEE!";     # For debugging
  sleep(1);                                                                   # For debugging

} # end sub slew_master

#-------------------------------------------------------#
sub slew_telescope
{
   my(@position) = @_;

   #Give coordinates as two digits - i.e., HH:MM:SS or DD:MM:SS so they can be read
   #by the gemini mount.  SS are rounded UP (55.8 -> 56).
   substr($position[0],0,0) = '0'x(2-length($position[0])); $RAh = $position[0];
   substr($position[1],0,0) = '0'x(2-length($position[1])); $RAm = $position[1];
   $RAs = int($position[2] + 0.5); substr($RAs,0,0) = '0'x(2-length($RAs));
   if ($sign < 0)
      { if ($position[3] == 0) {$position[3] = '-00'}
        else { $position[3] = abs($position[3]);
               substr($position[3],0,0) = '0'x(2-length($position[3]));
               $position[3] = '-'.$position[3];
              }
       }
   else {substr($position[3],0,0) = '0'x(2-length($position[3]))}
   $DECd = $position[3];
   $position[4] = abs($position[4]) if ($sign < 0 && $position[3] == 0);
   $position[5] = abs($position[5]) if ($sign < 0 && $position[3] == 0 && $position[4] == 0);
   substr($position[4],0,0) = '0'x(2-length($position[4])); $DECm = $position[4];
   $position[5] = int($position[5] + 0.5);
   substr($position[5],0,0) = '0'x(2-length($position[5])); $DECs = $position[5];
```

```
   $mount_error = 'none';          # Initialize variable
   $error = 0;                     # Initialize variable


  #  SEND_SLEW_COMMANDS:
    &SEND_RA;                       # Send RA coord. to Gemini
    &SEND_DEC;                      # Send DEC coord. to Gemini
    &send_name;                     # Send name of field to gemini

} # end sub slew_telescope
```

Appendix D

The PBASIC program: focus_filter_dewar3.bs2

This program is running on the BASIC Stamp which operates the focusing mechanism, the filter wheel and the LN_2 refill system.

```
'focus_filter_dewar2.bs2
'
'Created: 1/9/02       By: Ken Portock & Phillip Nelson
'
'Modified for use with one stepper motor.
'Modified 12/11/02
'
'For use with the PERL Programs:
'- focus_controller.pl
'- filter_controller.pl
'- fill_dewar_v4.pl
'
'Controls filter wheel, focus mechanism and dewar fill controller together.
'Uses stepper3.bs2, logictest.bs2 and mechvalve_v4.bs2 as subroutines
'
'Effectively, The PERL script calls the subroutines using these commands:
'PERL sends a "1" to initiate the focusing routine.
'PERL sends a "2" to initiate the filter wheel routine.
'PERL sends a "3" to initiate the dewar filling routine.
'

'=================================Main Program=======================================

command var byte ' variable set by PERL to designate
which subroutine to call

  currentpos =   0 ' Assume stepper is home at
start of program

Loop:

serin 16,16780,[DEC1 command] ' Receive command from PERL; 1, 2 or 3

if command = 1 then call_focus ' input of 1 from PERL goes to focus control
routine.
if command = 2 then call_filter  ' input of 2 from PERL goes to filter
control routine.
if command = 3 then call_dewar ' input of 3 from PERL goes to dewar fill
routine.
if command = 4 then call_init_focus ' input of 4 from PERL goes to
initialize focus routine.

goto Loop

call_focus
gosub Focus ' calls the focus controller.
goto Loop ' go back to command input line
when done.

call_filter
gosub Filter ' calls the filter controller.
goto Loop ' go back to command input line
when done.

call_dewar
gosub Dewar ' calls the dewar fill
controller.
goto Loop ' go back to command input line
when done.

call_init_focus
gosub init_focus ' calls the dewar fill
```

```
controller.
goto Loop ' go back to command input line
when done.


end

'============================Focus Control Routine================================
'
' Floppy stepper motor driver
'
' KPP 12/12/01
'
' modified stepper3.bs2
'
' Read motor number and 4-digit new position to rotate motor,
' New position can be changed to read 5 digits for a larger range of
' focusing operation. Must also edit steptest.pl to output 5 digits.
'
' Motor numbers modified to read a 1 or 2 to turn motors 0 or 1
' respectively to be compatable with PERL output.
'
' When new position = 0 then motor will go back to postition Zero
' This is modified from stepper1.bs2 and stepper2.bs2.
'
'
'
' BSII            Floppy
' Pin      Port
'  5    0     14          Select 2
'  6 1     12  Select 1
'  7    2     18          Direction
'  8    3     20          Step
' 20        15 26          Track 00 (pulled high)
' 23         Gnd         GND
'
'
init_focus:
currentpos var word ' Current step # the motor is located

serin 16,16780,[DEC4 currentpos] ' initialize curent position, number
given by SLICAR
return


Focus:
' Name I/O pins
driveDS CON 1    '0 ' Stepper select
driveDIR CON 2  ' Stepper direction
driveSTEP CON 3     ' Step pulse line
driveTRK0 CON 15    ' Drive track 0

steps var word ' # of steps to move stepper motor
newpos var word ' Position inpu from PERL that the stepper will
rotate to
i var word ' Counter

  output driveDS ' Configure I/O pins
  output driveDIR
  output driveSTEP
  input driveTRK0

  high driveSTEP
  high driveDS
```

```
  serin 16,16780,[DEC4 newpos] ' Get Get new position to turn to from PERL
  low driveDS ' Select drive
  gosub DoMove ' Call motor activation routine
  high driveDS ' Deselect controller
  serout 16,16780,[DEC4 newpos] ' Tell PERL when done and confirm new position
  return ' Return to main program

'-----------------Stepper Motor Activate Routine---------------------
DoMove:

  if newpos = currentpos then DoSteps_exit ' Don't move

  if newpos > currentpos then StepUp ' Move to a higher step position

  high driveDIR ' If newpos < currrentpos move to lower
step position
  steps = currentpos - newpos  ' Calculate # of steps
  goto DoSteps_move ' If newpos = 0, motor goes to position
0 (does NOT look for "home")
' Modified from stepper2.bs2 and
stepper1.bs2

StepUp:
  low driveDIR
  steps = newpos - currentpos ' Calculate # of steps

DoSteps_Move
  for i=1 to steps ' Loops # of steps to get to new
position
    pulsout driveSTEP,2000 ' Pulsout is steps the motor one step
  next

  currentpos = newpos ' Reset current position

DoSteps_exit ' Returns to Main
  return

'============================Filter Wheel Controller====================================

'copied logictest2.bs2
'KPP 12/13/01
'Used to control filter wheel
'
'INB is I/O pins 4 through 7
'Pins 4, 5, and 6 are LED display,(binary number, NOT NESSASARILY THE FILTER NUMBER)
'pin 7 is button output.  Pulse High to push button
'pin 4 : blue
'pin 5 : orange <-- these pins tell which filter its on
'pin 6 : gray
'pin 7 : brown     Drive high to turn filter wheel.


Filter:


check var byte
in_num var byte
filter_num var byte
filter_goal  var byte
missed var byte


serin 16, 16780,[DEC1 filter_goal] ' waits for PERL to tell Stamp the
```

```
desired filter number
missed = 0

Filter_top

LOW 7
in_num =  INB ' get binary value from LED display

if in_num = 3 then Filter_1 ' input of 3 corresponds to filter #1
if in_num = 4 then Filter_2 ' input of 4 corresponds to filter #2
if in_num = 0 then Filter_3 ' input of 0 corresponds to filter #3
if in_num = 2 then Filter_4 ' input of 2 corresponds to filter #4
if in_num = 7 then Filter_top ' input of 7 means filter is in transit,
check again

Filter_1
filter_num = 1
GOTO check_it
Filter_2
filter_num = 2
GOTO check_it
Filter_3 ' sets filter_num to the correct filter
filter_num = 3 ' that is engaged, then jumps to Check_it.
GOTO check_it
Filter_4
filter_num = 4
GOTO check_it

GOTO Filter_top

Check_it
if filter_num = filter_goal then Done ' Checks if on correct filter or
not.
if filter_num <> filter_goal then Turn_it ' If not the turn to the
next filter

Done
serout 16,16780,[DEC1 filter_num] ' confirm filter number with
PERL
return ' If on correct filter then stop
(return to main program)

Turn_it ' If not on correct filter then
advance
if missed >= 5 then filter_failed ' Check miss counter, stop if
missed 5 or more times
missed = missed + 1 ' increment miss counter
HIGH 7 ' activate filter motor,
advances to next filter
Pause 500 ' correct filter
LOW 7 ' deactivate filter
motor
Goto Filter_Top

filter_failed
LOW 7
      filter_num = missed                       ' if never reached filter then send a
zero to perl
Goto Done ' indicating that it
failed


'====================Dewar Filler Controller=====================
'{$STAMP BS2} 'Stamp directive: specifies this program is for a BS2
```

```
'
'
'PURPOSE:
'This program causes the solid state relay to close, opening the mechanical valve.
'While the camera dewar is being refilled, information is sent back to the PERL
'program to check the state of pin 10.  Once the camera dewar is full, the
'mechanical valve closes and the state of pin '10 flips from a 0 to 1.  This
'causes the program to open the relay again and allows observations to continue.
'
'CREATED: January 9, 2002 BY: Phillip Nelson
'Updated: July 10, 2002


Dewar: 'Label which indicates the beginning of the program

  fill_status var byte
  pin10  var byte        ' pin 10 variable

  fill_status = 1 'Default valve status - 1 = open; 2 = closed

  HIGH 8 'Causes relay to close (mechanical valve opens; LN2 flows)
'Pin 8 (P8) is set to an output and given the value '1'.
'P10 will have a value of 0 to begin.

  Open: 'Just a label
PAUSE 500 '0.5 second pause
pin10 = IN10 'get status of the sensor
SEROUT 16, 16780, [DEC1 pin10] 'Sends the state of pin 10 to PERL
SERIN 16, 16780, 100, Continue, [DEC1 fill_status]
'Wait for PERL to send valve status.  This is necessary if refill time goes
'beyond 1 hour.  PERL needs to tell valve to close and programs to reset.

      Continue:
IF fill_status = 2 THEN Fill_Done
IF pin10 = 0 THEN Open 'Keep valve open

  Fill_Done:
  LOW 8 'Once sensor has triggered the valve to close, keeps it closed.
'A cold sensor causes pin 10 (P10) to be set to a '1'; this
'command keeps P10 a 1 and opens the relay.

return 'Go back to start of program
```

Appendix E

The PBASIC: Roof_v7.bs2

This program is running on the BASIC Stamp which is controlling the roof system.

```
'{$STAMP BS2}
' Roof Control Program KPP 1/30/02
' last modified 7/25/02
' Modified roof_test_v3.bs2 to work with PERL
' PERL sends: 1 to open the roof, also resets counter
'        2 to close the roof, also resets counter
'    3 to reset counter
' Counter to check if PC is still 'ok'
' Each decrement is approx. 100millisecinds
' After (time x 100)ms, STAMP times out, and closes roof.
' Unless counter is reset.

' Pin configuration:
' P1 yellow roof power.  high is power on.
' P2 gray roof direction.  low is open
'
' P3 violet manual switch, open;  low is open, high is nothing.
' P4 brown manual switch, close;  low is close, high is nothing.

' P5  limit switch, stop opening when low.
' P6  limit switch, stop closing when low.

' P10              power detection relay   = 0 when POWER IS CONNECTED
' = 1 when POWER IS DISCONNECTED
' BOTTOM relay is open
' TOP relay is close
'roof_dir = 0 ' flips BOTTOM relay -> opens roof
'roof_pwr = 1

'roof_dir = 1 ' flips TOP relay -> closes roof
'roof_pwr = 1


command var word ' command variable 1 for open, 2 for close
counter var word ' command time out counter
counter2 var word ' open/close timeout counter
time var word ' time for command time out counter
time2 var word ' time for open/close timer
status var word ' knows status of roof; open(=1) or closed(=2)
check var word ' sent to perl to check if command has been followed

roof_pwr var OUT1 ' pin to enable roof power
roof_dir var OUT2 ' pin to set roof direction
sw_open var IN3 ' manual switch, open
sw_close var IN4 ' manual switch, close
open_lim var IN8 ' limit switch to stop opening
close_lim    var IN9 ' limit switch to stop closing
check_power var IN10  ' pin that watches for power failure
auto_dial var OUT11 ' pin that activates autodialer for emergencies



'     5432109876543210
DIRS=%0000100000000110


command = 0
time = 12000 ' command wait time, corresponds to wait time of approx. 20 minutes, 10 = 1 sec
time2 = 3000 ' open/close wait time, corresponds to wait time of approx. 5 minutes, 10 = 1 sec


Start:
  counter = time ' command received, reset the counter
```

```
'Waiting for Command:  press 1 to open, 2 to close, 3 to reset counter (PC ' I'm Okay ')
Wait_command:

roof_pwr = 0 ' disable roof power
roof_dir = 0
IF sw_close = 0 THEN CLOSE ' close roof if manual switch is switched to close
IF sw_open = 0 THEN OPEN ' open roof if manual switch is switched to open

    IF check_power = 1 then POWER_OUT ' check power, close if power is out
counter = counter - 1 ' decrement counter by 1, 100ms wait time
IF counter = 0 then TIME_OUT

SERIN 16,16780,100,Wait_command,[DEC1 command] ' wait 100 milliseconds for input command, after 100ms start over
IF command = 1 then OPEN ' open roof
IF command = 2 then CLOSE ' close roof
IF command = 3 then CHECK_STATUS ' PERL asked for a status check, send current status, RESET COUNTER

GOTO Wait_command


CHECK_STATUS: '-------------------- Status Check ----------------------- '
IF open_lim = 0 THEN SEND_OPEN ' check open limit switch
IF close_lim = 0 THEN SEND_CLOSE ' check close limit switch
     GOTO SEND_STATUS ' send status

SEND_OPEN
status = 2 ' set status to "roof is open" mode
    GOTO SEND_STATUS ' send status

SEND_CLOSE
status = 4 ' set status to "roof is closed" mode
    GOTO SEND_STATUS ' send status


SEND_STATUS
SEROUT 16,16780,[DEC1 status] ' tell PERL the current roof status
    GOTO start   ' go to beginning (reset the counter)


OPEN:     '-------------------- Open ----------------------- '
roof_pwr= 0 ' disable roof power
roof_dir = 0 ' set direction to open
roof_pwr = 1 ' enable roof power
status = 1 ' set status to "roof is opening" mode
counter2 = time2 ' reset open/close counter
    OPENING
IF open_lim = 0  THEN HALT_OPEN      ' stop when limit switch is reached
IF sw_close = 0  THEN WAITCLOSE        ' stop and reverse if manual switch is switched to close
PAUSE 100 ' wait 100 milliseconds
counter2 = counter2 - 1 ' decrement counter2
IF counter2 = 0 THEN PHONE_HOME ' Never reached limit switch, call home
    GOTO OPENING

HALT_OPEN:
roof_pwr = 0 ' disable roof power
status = 2 ' set status to "roof is open" mode
    GOTO SEND_STATUS ' return to beginning, reset counter

WAITCLOSE
      roof_pwr = 0 ' disable roof power
      roof_dir = 0
status = 3 ' set status to "roof is closing" mode
PAUSE 2000
    GOTO CLOSE
```

```
CLOSE:     '-------------------- Close ------------------------ '
roof_pwr= 0 ' disable roof power
roof_dir = 1 ' set direction to close
roof_pwr = 1 ' enable roof power
counter2 = time2 ' reset open/close counter
    CLOSING
IF close_lim = 0  THEN HALT_CLOSE       ' stop when limit switch is reached
IF sw_open = 0    THEN WAITOPEN        ' stop and reverse if manual switch is switched to close
PAUSE 100 ' wait 100 milliseconds
counter2 = counter2 - 1 ' decrement counter2
IF counter2 = 0 THEN PHONE_HOME ' Never reached limit switch, call home
    GOTO CLOSING

HALT_CLOSE:
roof_pwr = 0 ' disable roof power
status = 4 ' set status to "roof is closed" mode
GOTO SEND_STATUS ' return to beginning, reset ccounter

WAITOPEN
      roof_pwr = 0 ' disable roof power
      roof_dir = 0 ' pull direction bit low
PAUSE 2000
    GOTO OPEN

' ----------------------- Emergency closing ---------------------- '

POWER_OUT:
status = 7 ' set status to power out
GOTO EMERGENCY_CLOSE ' close roof

TIME_OUT:
status = 5 ' set ststus to time out
GOTO EMERGENCY_CLOSE ' close roof

EMERGENCY_CLOSE:

roof_pwr= 0 ' disable roof power
roof_dir = 1 ' set direction to close
roof_pwr = 1 ' enable roof power

counter2 = time2 ' reset open/close counter
EMERG_CLOSE_LOOP:
IF close_lim = 0 THEN EMERG_CLOSE_HALT             ' stop when limit switch is reached
PAUSE 100 ' wait 100 milliseconds
counter2 = counter2 - 1 ' decrement counter2
IF counter2 = 0 THEN PHONE_HOME ' Never reached limit switch, call home
SERIN 16, 16780,100,EMERG_CLOSE_LOOP,[DEC1 command]   ' check for messages from PERL
SEROUT 16,16780,[DEC1 status] ' send status to perl
    GOTO EMERG_CLOSE_LOOP

EMERG_CLOSE_HALT:
roof_pwr = 0                   ' disable roof power
SERIN 16, 16780,1000,EMERG_CLOSE_HALT,[DEC1 command]  ' check for messages from PERL
SEROUT 16,16780,[DEC1 status]     ' tell PERL when done, and status
         GOTO EMERG_CLOSE_HALT


end

' ------------------------ PHONE HOME --------------------------- '
PHONE_HOME:
status = 8 ' set status to phone home
```

```
auto_dial = 1 ' activate autodialer
SEROUT 16,16780,[DEC1 status]              ' tell PERL the current roof status
PAUSE 30000 ' wait 30 seconds
auto_dial = 0 ' deactivate phone dialer
      PAUSE 60000
PAUSE 60000
      PAUSE 60000
PAUSE 60000
      PAUSE 60000
PAUSE 60000
      PAUSE 60000
PAUSE 60000
      PAUSE 60000
PAUSE 60000
   GOTO PHONE_HOME
```

Appendix F

PMIS Macro: startnew.cmd

A PMIS macro used by SLICAR when communicating with PMIS.

```
flag echo off
* FITSNEW.CMD
*
* Outputs a 16-bit FITS file with useful keywords.
* Image values are shifted down by 32768, so high intensity pixels
* will have higher integer values than low intensity pixels (no wrap-around
* along intensity axis).  FITS values BZERO and BSCALE will make sure
* that correct pixel values (between 0 and 2**16) result when the FITS file
* is read into AIPS or IRAF.
*
* Runs in conjunction with Perl program: savefits.pl.
*
* Edited FITS.CMD.  All previously user inputted variables are
* now inputted into PMIS from PERL.
* The remainder of the macro remains unchanged from the original.
* This macro is called from the perl program  savefits.pl.
*
* startmacro.pl (and therefore startnew.cmd) must be run before this
* macro is run.
*
* K.P. Portock  5/16/02


  vdefine igain o.gain
  if @igain - 4
    vdefine egain "10.44"
    vdefine rdnoise "14.69"
  else
    vdefine egain "2.53"
    vdefine rdnoise "6.9"
  endif
  vdefine msec o.time
  vdefine secs @msec/1000.0
  vdefine datim i.mtime
  string word 1 @datim
  vdefine date @tvalue
  string word 2 @datim
  vdefine time @tvalue
  fitscard clear
  fitscard observat str @observ
  fitscard telescop str @tele
  fitscard instrume str @lens
```

```
fitscard object str   @object
fitscard date-obs str @date
fitscard time-obs str @time
fitscard imagetyp str @type
fitscard filter int   @ifilter
fitscard exptime real @secs
fitscard gain real    @egain
fitscard rdnoise real @rdnoise
fitscard camtemp real @ctemp
i- 32768
fitscard bzero real 32768.0
fitscard bscale real 1.0
export fits @fname
```

Appendix G

PMIS Macro: fitsnew.cmd.bs2

A PMIS macro used by SLICAR when communicating with PMIS.

```
flag echo off
* FITSNEW.CMD
*
* Outputs a 16-bit FITS file with useful keywords.
* Image values are shifted down by 32768, so high intensity pixels
* will have higher integer values than low intensity pixels (no wrap-around
* along intensity axis).  FITS values BZERO and BSCALE will make sure
* that correct pixel values (between 0 and 2**16) result when the FITS file
* is read into AIPS or IRAF.
*
* Runs in conjunction with Perl program: savefits.pl.
*
* Edited FITS.CMD.  All previously user inputted variables are
* now inputted into PMIS from PERL.
* The remainder of the macro remains unchanged from the original.
* This macro is called from the perl program  savefits.pl.
*
* startmacro.pl (and therefore startnew.cmd) must be run before this
* macro is run.
*
* K.P. Portock  5/16/02


  vdefine igain o.gain
  if @igain - 4
    vdefine egain "10.44"
    vdefine rdnoise "14.69"
  else
    vdefine egain "2.53"
    vdefine rdnoise "6.9"
  endif
  vdefine msec o.time
  vdefine secs @msec/1000.0
  vdefine datim i.mtime
  string word 1 @datim
  vdefine date @tvalue
  string word 2 @datim
  vdefine time @tvalue
  fitscard clear
  fitscard observat str @observ
  fitscard telescop str @tele
  fitscard instrume str @lens
```

```
fitscard object str   @object
fitscard date-obs str @date
fitscard time-obs str @time
fitscard imagetyp str @type
fitscard filter int   @ifilter
fitscard exptime real @secs
fitscard gain real    @egain
fitscard rdnoise real @rdnoise
fitscard camtemp real @ctemp
i- 32768
fitscard bzero real 32768.0
fitscard bscale real 1.0
export fits @fname
```

Appendix H

An Example Observing File

```
Homer Jay Simpson
FLAT
172900   380203   one  1  5  3
172900   380203   one  3  5  3
125209   045321   two  2  5  3
160654   370950   three  3  5   3
160654   370950   three  2  5   3
130702   630523   four  4  5   3
095152   514726   five  1  5   3 EAST
FILL
123043   132522   six  2  4   6
161938   -090925   seven  3  4   6
151912   -140625   eight  4  4   6
165415   691421   nine  1  4   6
155512   071948   ten  2  4   6
155512   071948   ten  4  4   6
154224   671457   elevn  3  4   6
144841   433122   twelv  4  4   6 EAST
135037   -022044   thrtn  1  4   6
```

Appendix I

An Example Log File

```
Polaris monitor started Mon Nov  4 12:23:48 2002 EST.

OBSERVING LOG - 02_11_04
========================

OBSERVER:  Homer Jay Simpson


Actual observing begins approximately at PM TWILIGHT which occurs at 23h 50m UT.
Observing will end by MOONRISE which occurs at 05h 05m UT.
MAXIMUM OBSERVING TIME: 05h 15m.

Mon Nov  4 19:33:13 2002 UT- Camera cool-down in progress...
Mon Nov  4 19:35:13 2002 UT- Camera cool-down complete...


Mon Nov  4 19:35:14 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias01.fit.

Mon Nov  4 19:35:36 2002 UT- Bias image saved as bias01.fit.

Mon Nov  4 19:35:37 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias02.fit.

Mon Nov  4 19:35:59 2002 UT- Bias image saved as bias02.fit.

Mon Nov  4 19:36:00 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias03.fit.

Mon Nov  4 19:36:22 2002 UT- Bias image saved as bias03.fit.

Mon Nov  4 19:36:23 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias04.fit.

Mon Nov  4 19:36:45 2002 UT- Bias image saved as bias04.fit.

Mon Nov  4 19:36:46 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias05.fit.

Mon Nov  4 19:37:09 2002 UT- Bias image saved as bias05.fit.

Mon Nov  4 19:37:10 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias06.fit.

Mon Nov  4 19:37:32 2002 UT- Bias image saved as bias06.fit.

Mon Nov  4 19:37:33 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias07.fit.

Mon Nov  4 19:37:55 2002 UT- Bias image saved as bias07.fit.

Mon Nov  4 19:37:56 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias08.fit.

Mon Nov  4 19:38:19 2002 UT- Bias image saved as bias08.fit.

Mon Nov  4 19:38:20 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias09.fit.

Mon Nov  4 19:38:42 2002 UT- Bias image saved as bias09.fit.

Mon Nov  4 19:38:43 2002 UT- Capturing bias image (0 sec. exposure). Image saved to: bias10.fit.

Mon Nov  4 19:39:05 2002 UT- Bias image saved as bias10.fit.

Mon Nov  4 19:39:05 2002 UT- one is in the East - HA = -0.0640162296745758.
Mon Nov  4 19:39:05 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 19:39:05 2002 UT- Slewing telescope to (17:29:00, 38:02:03).

Mon Nov  4 19:40:45 2002 UT- Turning filter wheel...
Mon Nov  4 19:40:45 2002 UT- filter = 1
```

```
Mon Nov  4 19:40:45 2002 UT- Focuser turning to postion = 0808
Mon Nov  4 19:40:49 2002 UT- New Postion 0808 confirmed.
Mon Nov  4 19:40:49 2002 UT- Capturing image 1/1 of one.
FIELD NAME: one
(RA, DEC) = (17h 29m 00s, 38d 02' 03")
(ALT, AZ) = (87d 00' 47", 75d 14' 51")
LST = 17:14:19   HA = 00:-14:40
FILTER #: 1   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 19:41:16 2002 UT- Image exposure and transfer complete.
Image saved to: 00rone.fit.

Mon Nov  4 19:41:19 2002 UT- Slewing telescope to (11:26:07, 01:49:07).


Mon Nov  4 19:42:55 2002 UT- Dewar refill begun.
Mon Nov  4 19:44:21 2002 UT- Dewar refill complete...


Mon Nov  4 19:44:22 2002 UT- Slewing telescope to (11:39:36, 87:36:41).

Mon Nov  4 19:45:44 2002 UT- Returning to home position...
Mon Nov  4 19:45:44 2002 UT- Slewing telescope to (11:40:58, 87:36:41).

Mon Nov  4 19:45:57 2002 UT- Home find complete...



Mon Nov  4 19:45:57 2002 UT- two is in the East - HA = 1.17401329617535.
Mon Nov  4 19:45:57 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 19:45:57 2002 UT- * * * UNABLE TO VIEW THIS FIELD AT THIS TIME * * *
Mon Nov  4 19:45:57 2002 UT- Altitude is less than 35.0 degrees.
FIELD NAME: two
(RA, DEC) = (12h 52m 09s, 04d 53' 21")
(ALT, AZ) = (20d 58' 06", 259d 46' 37")
LST = 17:21:12   HA = 04:29:03
FILTER #: 2   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 19:45:57 2002 UT- Returning to home position...
Mon Nov  4 19:45:57 2002 UT- Slewing telescope to (11:41:11, 87:36:41).

Mon Nov  4 19:46:08 2002 UT- Home find complete...



Mon Nov  4 19:46:08 2002 UT- three is in the East - HA = 0.325058249354417.
Mon Nov  4 19:46:08 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 19:46:08 2002 UT- Slewing telescope to (16:06:54, 37:09:50).

Mon Nov  4 19:47:41 2002 UT- Turning filter wheel...
Mon Nov  4 19:47:55 2002 UT- filter = 3
Mon Nov  4 19:47:55 2002 UT- Focuser turning to postion = 2560
Mon Nov  4 19:48:04 2002 UT- New Postion 2560 confirmed.
Mon Nov  4 19:48:04 2002 UT- Capturing image 1/1 of three.
FIELD NAME: three
(RA, DEC) = (16h 06m 54s, 37d 09' 50")
(ALT, AZ) = (75d 11' 50", 275d 01' 53")
LST = 17:21:23   HA = 01:14:29
FILTER #: 3   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 19:48:31 2002 UT- Image exposure and transfer complete.
Image saved to: 01dthree.fit.

Mon Nov  4 19:48:33 2002 UT- Returning to home position...
Mon Nov  4 19:48:33 2002 UT- Slewing telescope to (11:43:47, 87:36:41).
```

```
Mon Nov  4 19:49:45 2002 UT- Home find complete...



Mon Nov  4 19:49:45 2002 UT- four is in the East - HA = 1.12569852773501.
Mon Nov  4 19:49:45 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 19:49:45 2002 UT- Slewing telescope to (13:07:02, 63:05:23).

Mon Nov  4 19:50:18 2002 UT- Turning filter wheel...
Mon Nov  4 19:50:25 2002 UT- filter = 4
Mon Nov  4 19:50:25 2002 UT- Focuser turning to postion = 8701
Mon Nov  4 19:50:55 2002 UT- New Postion 8701 confirmed.
Mon Nov  4 19:50:55 2002 UT- Capturing image 1/1 of four.
FIELD NAME: four
(RA, DEC) = (13h 07m 02s, 63d 05' 23")
(ALT, AZ) = (44d 04' 55", 325d 20' 34")
LST = 17:25:01   HA = 04:18:00
FILTER #: 4   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 19:51:22 2002 UT- Image exposure and transfer complete.
Image saved to: 02sfour.fit.

Mon Nov  4 19:51:25 2002 UT- Returning to home position...
Mon Nov  4 19:51:25 2002 UT- Slewing telescope to (11:46:40, 87:36:41).

Mon Nov  4 19:52:04 2002 UT- Home find complete...



Mon Nov  4 19:52:04 2002 UT- five is in the East - HA = 1.98740979963924.
Mon Nov  4 19:52:04 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 19:52:04 2002 UT- * * * UNABLE TO VIEW THIS FIELD AT THIS TIME * * *
Mon Nov  4 19:52:04 2002 UT- Altitude is less than 35.0 degrees.
FIELD NAME: five
(RA, DEC) = (09h 51m 52s, 51d 47' 26")
(ALT, AZ) = (16d 06' 30", 323d 55' 54")
LST = 17:27:20   HA = 07:35:28
FILTER #: 1   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 19:52:04 2002 UT- Returning to home position...
Mon Nov  4 19:52:04 2002 UT- Slewing telescope to (11:47:19, 87:36:41).

Mon Nov  4 19:52:17 2002 UT- Home find complete...



Mon Nov  4 19:52:17 2002 UT- Slewing telescope to (11:37:07, 01:49:07).


Mon Nov  4 19:53:38 2002 UT- Dewar refill begun.
Mon Nov  4 19:54:55 2002 UT- Dewar refill complete...


Mon Nov  4 19:54:56 2002 UT- Slewing telescope to (11:50:11, 87:36:41).

Mon Nov  4 19:56:19 2002 UT- six is in the East - HA = 1.31289081586552.
Mon Nov  4 19:56:19 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 19:56:19 2002 UT- * * * UNABLE TO VIEW THIS FIELD AT THIS TIME * * *
Mon Nov  4 19:56:19 2002 UT- Altitude is less than 35.0 degrees.
FIELD NAME: six
(RA, DEC) = (12h 30m 43s, 13d 25' 22")
(ALT, AZ) = (19d 45' 26", 272d 04' 44")
```

```
LST = 17:31:36   HA = 05:00:53
FILTER #: 2   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 19:56:19 2002 UT- Returning to home position...
Mon Nov  4 19:56:19 2002 UT- Slewing telescope to (11:51:34, 87:36:41).

Mon Nov  4 19:56:37 2002 UT- Home find complete...



Mon Nov  4 19:56:37 2002 UT- seven is in the East - HA = 0.315366010213096.
Mon Nov  4 19:56:37 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 19:56:38 2002 UT- Slewing telescope to (16:19:38, -09:09:25).

Mon Nov  4 19:58:07 2002 UT- Turning filter wheel...
Mon Nov  4 19:58:28 2002 UT- filter = 3
Mon Nov  4 19:58:28 2002 UT- Focuser turning to postion = 2560
Mon Nov  4 19:58:59 2002 UT- New Postion 2560 confirmed.
Mon Nov  4 19:58:59 2002 UT- Capturing image 1/1 of seven.
FIELD NAME: seven
(RA, DEC) = (16h 19m 38s, -9d 09' 25")
(ALT, AZ) = (40d 31' 36", 203d 45' 24")
LST = 17:31:54   HA = 01:12:16
FILTER #: 3   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 19:59:26 2002 UT- Image exposure and transfer complete.
Image saved to: 03dseven.fit.

Mon Nov  4 19:59:28 2002 UT- Returning to home position...
Mon Nov  4 19:59:28 2002 UT- Slewing telescope to (11:54:44, 87:36:41).

Mon Nov  4 20:01:00 2002 UT- Home find complete...



Mon Nov  4 20:01:00 2002 UT- eight is in the East - HA = 0.598234436061318.
Mon Nov  4 20:01:00 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 20:01:00 2002 UT- * * * UNABLE TO VIEW THIS FIELD AT THIS TIME * * *
Mon Nov  4 20:01:00 2002 UT- Altitude is less than 35.0 degrees.
FIELD NAME: eight
(RA, DEC) = (15h 19m 12s, -14d 06' 25")
(ALT, AZ) = (29d 18' 15", 218d 46' 53")
LST = 17:36:18   HA = 02:17:06
FILTER #: 4   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 20:01:00 2002 UT- Returning to home position...
Mon Nov  4 20:01:00 2002 UT- Slewing telescope to (11:56:16, 87:36:41).

Mon Nov  4 20:01:20 2002 UT- Home find complete...



Mon Nov  4 20:01:20 2002 UT- nine is in the East - HA = 0.184958995726718.
Mon Nov  4 20:01:20 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 20:01:20 2002 UT- Slewing telescope to (16:54:15, 69:14:21).

Mon Nov  4 20:02:41 2002 UT- Turning filter wheel...
Mon Nov  4 20:02:55 2002 UT- filter = 1
Mon Nov  4 20:02:55 2002 UT- Focuser turning to postion = 0808
Mon Nov  4 20:03:04 2002 UT- New Postion 0808 confirmed.
Mon Nov  4 20:03:04 2002 UT- Capturing image 1/1 of nine.
FIELD NAME: nine
(RA, DEC) = (16h 54m 15s, 69d 14' 21")
(ALT, AZ) = (57d 34' 24", 353d 01' 01")
```

```
LST = 17:36:38   HA = 00:42:23
FILTER #: 1   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 20:03:31 2002 UT- Image exposure and transfer complete.
Image saved to: 04rnine.fit.

Mon Nov  4 20:03:33 2002 UT- Returning to home position...
Mon Nov  4 20:03:33 2002 UT- Slewing telescope to (11:58:50, 87:36:41).

Mon Nov  4 20:04:54 2002 UT- Home find complete...



Mon Nov  4 20:04:54 2002 UT- ten is in the East - HA = 0.458218354477278.
Mon Nov  4 20:04:54 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 20:04:54 2002 UT- Slewing telescope to (15:55:12, 07:19:48).

Mon Nov  4 20:06:14 2002 UT- Turning filter wheel...
Mon Nov  4 20:06:21 2002 UT- filter = 2
Mon Nov  4 20:06:21 2002 UT- Focuser turning to postion = 4253
Mon Nov  4 20:06:38 2002 UT- New Postion 4253 confirmed.
Mon Nov  4 20:06:38 2002 UT- Capturing image 1/1 of ten.
FIELD NAME: ten
(RA, DEC) = (15h 55m 12s, 07d 19' 48")
(ALT, AZ) = (51d 41' 24", 225d 03' 03")
LST = 17:40:12   HA = 01:45:00
FILTER #: 2   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 20:07:05 2002 UT- Image exposure and transfer complete.
Image saved to: 05hten.fit.

Mon Nov  4 20:07:07 2002 UT- Returning to home position...
Mon Nov  4 20:07:07 2002 UT- Slewing telescope to (12:02:24, 87:36:41).

Mon Nov  4 20:08:28 2002 UT- Home find complete...



Mon Nov  4 20:08:28 2002 UT- elevn is in the East - HA = 0.529674018465993.
Mon Nov  4 20:08:28 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 20:08:28 2002 UT- Slewing telescope to (15:42:24, 67:14:57).

Mon Nov  4 20:09:32 2002 UT- Turning filter wheel...
Mon Nov  4 20:09:39 2002 UT- filter = 3
Mon Nov  4 20:09:39 2002 UT- Focuser turning to postion = 2560
Mon Nov  4 20:09:48 2002 UT- New Postion 2560 confirmed.
Mon Nov  4 20:09:48 2002 UT- Capturing image 1/1 of elevn.
FIELD NAME: elevn
(RA, DEC) = (15h 42m 24s, 67d 14' 57")
(ALT, AZ) = (55d 32' 50", 339d 47' 40")
LST = 17:43:47   HA = 02:01:23
FILTER #: 3   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 20:10:15 2002 UT- Image exposure and transfer complete.
Image saved to: 06delevn.fit.

Mon Nov  4 20:10:17 2002 UT- Returning to home position...
Mon Nov  4 20:10:17 2002 UT- Slewing telescope to (12:05:35, 87:36:41).

Mon Nov  4 20:11:19 2002 UT- Home find complete...



Mon Nov  4 20:11:19 2002 UT- twelv is in the East - HA = 0.77652671070661.
Mon Nov  4 20:11:19 2002 UT- Waiting for it to cross meridian.
```

```
Mon Nov  4 20:11:19 2002 UT- Slewing telescope to (14:48:41, 43:31:22).

Mon Nov  4 20:12:09 2002 UT- Turning filter wheel...
Mon Nov  4 20:12:16 2002 UT- filter = 4
Mon Nov  4 20:12:16 2002 UT- Focuser turning to postion = 8701
Mon Nov  4 20:12:46 2002 UT- New Postion 8701 confirmed.
Mon Nov  4 20:12:46 2002 UT- Capturing image 1/1 of twelv.
FIELD NAME: twelv
(RA, DEC) = (14h 48m 41s, 43d 31' 22")
(ALT, AZ) = (55d 59' 06", 294d 43' 03")
LST = 17:46:39   HA = 02:57:58
FILTER #: 4   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 20:13:13 2002 UT- Image exposure and transfer complete.
Image saved to: 07stwelv.fit.

Mon Nov  4 20:13:16 2002 UT- Returning to home position...
Mon Nov  4 20:13:16 2002 UT- Slewing telescope to (12:08:34, 87:36:41).

Mon Nov  4 20:14:05 2002 UT- Home find complete...



Mon Nov  4 20:14:05 2002 UT- thrtn is in the East - HA = 1.04199525277003.
Mon Nov  4 20:14:05 2002 UT- Waiting for it to cross meridian.

Mon Nov  4 20:14:05 2002 UT- * * * UNABLE TO VIEW THIS FIELD AT THIS TIME * * *
Mon Nov  4 20:14:05 2002 UT- Altitude is less than 35.0 degrees.
FIELD NAME: thrtn
(RA, DEC) = (13h 50m 37s, -2d 20' 44")
(ALT, AZ) = (22d 05' 10", 248d 35' 32")
LST = 17:49:25   HA = 03:58:48
FILTER #: 1   EXPOSURE TIME: 0.1 mins.
Mon Nov  4 20:14:05 2002 UT- Returning to home position...
Mon Nov  4 20:14:05 2002 UT- Slewing telescope to (12:09:23, 87:36:41).

Mon Nov  4 20:14:17 2002 UT- Home find complete...



Mon Nov  4 20:14:17 2002 UT-   * * * END OF NORMAL OBSERVING SESSION * * *


Mon Nov  4 20:14:17 2002 UT- Sytem shut down in progress....

Mon Nov  4 20:14:17 2002 UT- Focuser turning to postion = 0000
Mon Nov  4 20:15:01 2002 UT- New Postion 0000 confirmed.
Mon Nov  4 20:15:01 2002 UT- Parking telescope.
Mon Nov  4 20:15:01 2002 UT- Slewing telescope to (12:10:20, 87:36:41).

Mon Nov  4 20:15:13 2002 UT- Telescope park complete.

Mon Nov  4 20:15:13 2002 UT- Closing PMIS applications.
Mon Nov  4 20:15:17 2002 UT- PMIS shut-down complete.

Mon Nov  4 20:15:17 2002 UT- Copying OBS file, C:\SLIC\02_11_04_OBS.txt, to C:\SLIC\DATA\02_11_04.
Mon Nov  4 20:15:17 2002 UT- C:\SLIC\02_11_04_OBS.txt copy successful.
```

Appendix J

Moon Rise and Set Table

This moon rise and set table is obtained from the US Naval Observatory website:

http://aa.usno.navy.mil/data/docs/RS_OneYear.html.

The table obtained from the website is for an entire year, this sample is only from January to May 2002.

Astronomical Applications Dept.
U. S. Naval Observatory
Washington, DC 20392-5420

Location: W080 34, N37 20
Zone: 5h West of Greenwich

	Jan.		Feb.		Mar.		Apr.		May	
Day	Rise	Set	Rise	Set	Rise	Set	Rise	Set	Rise	Set
	h m	h m	h m	h m	h m	h m	h m	h m	h m	h m
01	1953	0940	2224	1007	2115	0835	2337	0849		0902
02	2106	1024	2334	1038	2227	0907		0933	0024	1000
03	2217	1102		1110	2337	0941	0042	1021	0111	1059
04	2327	1135	0042	1143		1016	0140	1115	0151	1159
05		1206	0149	1218	0045	1056	0231	1212	0225	1258
06	0035	1237	0254	1258	0150	1140	0314	1310	0255	1356
07	0143	1308	0356	1343	0251	1229	0351	1409	0321	1453
08	0249	1341	0454	1433	0345	1322	0423	1507	0346	1550
09	0355	1418	0547	1527	0432	1419	0451	1604	0410	1646
10	0500	1459	0632	1625	0513	1517	0517	1701	0435	1744
11	0602	1546	0712	1724	0549	1616	0542	1757	0502	1844
12	0659	1638	0746	1822	0619	1713	0606	1854	0531	1945
13	0750	1735	0816	1920	0647	1810	0632	1952	0605	2048
14	0835	1833	0843	2016	0713	1907	0659	2052	0645	2150
15	0913	1932	0909	2113	0737	2003	0730	2153	0731	2249
16	0945	2030	0934	2209	0802	2100	0805	2255	0826	2344
17	1014	2128	0959	2306	0828	2158	0846	2355	0927	
18	1041	2224	1025		0856	2258	0935		1034	0032
19	1106	2320	1055	0005	0928	2359	1032	0053	1143	0114
20	1131		1129	0105	1005		1135	0146	1254	0151
21	1157	0017	1209	0208	1049	0100	1244	0233	1404	0225
22	1225	0116	1258	0311	1141	0201	1356	0314	1515	0256
23	1257	0216	1356	0412	1242	0258	1508	0351	1626	0327
24	1335	0319	1502	0510	1350	0351	1621	0424	1739	0358
25	1420	0424	1615	0602	1503	0437	1735	0456	1851	0433
26	1515	0529	1731	0647	1618	0518	1849	0529	2002	0512
27	1619	0631	1847	0726	1733	0555	2002	0603	2110	0557
28	1730	0727	2002	0802	1848	0629	2115	0640	2211	0648
29	1844	0815			2002	0701	2224	0722	2303	0744
30	1959	0857			2115	0735	2328	0809	2348	0845
31	2113	0934			2228	0810				0946

Appendix K

Astronomical Twilight Table

This astronomical twilight table is obtained from the US Naval Observatory website:

http://aa.usno.navy.mil/data/docs/RS_OneYear.html.

The table obtained from the website is for an entire year, this sample is only from January to May 2002.

U. S. Naval Observatory
Astronomical Applications Dept.
Washington, DC 20392-5420

Location: W080 34, N37 20

Zone: 5h West of Greenwich

	Jan.		Feb.		Mar.		Apr.		May	
Day	Begin	End	Begin	End	Begin	End	Begin	End	Begin	End
	h m	h m	h m	h m	h m	h m	h m	h m	h m	h m
01	0603	1849	0556	1916	0527	1943	0439	2014	0350	2050
02	0603	1850	0555	1917	0526	1944	0437	2015	0348	2051
03	0604	1850	0555	1918	0524	1945	0436	2016	0347	2053
04	0604	1851	0554	1919	0523	1946	0434	2017	0345	2054
05	0604	1852	0553	1920	0522	1947	0432	2018	0344	2055
06	0604	1853	0552	1921	0520	1948	0431	2020	0342	2057
07	0604	1853	0552	1922	0519	1949	0429	2021	0341	2058
08	0604	1854	0551	1923	0517	1950	0427	2022	0339	2059
09	0604	1855	0550	1924	0516	1951	0426	2023	0338	2100
10	0604	1856	0549	1925	0514	1952	0424	2024	0337	2102
11	0604	1857	0548	1926	0513	1952	0422	2025	0335	2103
12	0604	1858	0547	1927	0511	1953	0421	2026	0334	2104
13	0604	1858	0546	1928	0510	1954	0419	2028	0333	2106
14	0604	1859	0545	1929	0508	1955	0417	2029	0331	2107
15	0604	1900	0544	1929	0507	1956	0416	2030	0330	2108
16	0603	1901	0543	1930	0505	1957	0414	2031	0329	2109
17	0603	1902	0542	1931	0504	1958	0412	2032	0328	2111
18	0603	1903	0541	1932	0502	1959	0411	2034	0326	2112
19	0603	1904	0540	1933	0500	2000	0409	2035	0325	2113
20	0602	1905	0538	1934	0459	2001	0407	2036	0324	2114
21	0602	1906	0537	1935	0457	2002	0406	2037	0323	2116
22	0602	1907	0536	1936	0456	2003	0404	2039	0322	2117
23	0601	1908	0535	1937	0454	2004	0402	2040	0321	2118
24	0601	1908	0534	1938	0452	2006	0401	2041	0320	2119
25	0600	1909	0532	1939	0451	2007	0359	2042	0319	2120
26	0600	1910	0531	1940	0449	2008	0358	2044	0318	2121
27	0559	1911	0530	1941	0447	2009	0356	2045	0317	2122
28	0559	1912	0528	1942	0446	2010	0354	2046	0316	2124
29	0558	1913			0444	2011	0353	2047	0315	2125
30	0558	1914			0442	2012	0351	2049	0315	2126
31	0557	1915			0441	2013			0314	2127

Vita

Kenneth P. Portock, the son of John and Margaret Portock, step-son to Elaine Portock, was born February 3, 1977, in Sellersville Pennsylvania. He graduated from Souderton Area High School in May 1995. He attended Drexel University in Philadelphia Pennsylvania and graduated with First Honors in June 2000, with his B.S. degree in Physics. This thesis completes his M.S. degree in Physics from Virginia Polytechnic Institute and State University.