Robinson Observatory Scale Model Project (November, 2019)

Anthony J. Eubanks, Brian T. Glass, Melinda I. Ramos, and Thomas A. Vilan

Dept. of Electrical and Computer Engineering (ECE), University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — This article presents a summary of the Robinson Observatory scale model project, which began in January, 2019 and concluded in November, 2019. The introduction to this paper includes a brief history of the Robinson Observatory as well as the primary motivations and goals of this project. The layout of the as-is system at the Observatory is considered alongside a high-level overview of our own design. Design decisions are addressed, to include component selection and software design. Finally, the challenges and successes of integration are summarized, along with a broad summary of the project and the bibliographies of all team members.**

***Index Terms* — Analog-digital Integrated Circuits, DC Motors, Differential Amplifiers, Microcontrollers, Optical Switches, System Integration, Telescopes, Universal Serial Bus.**

I. Introduction

Although the groundbreaking for the Robinson Observatory occurred in January of 1994, the story of our current telescope begins in 2007. It was at this point that the existing 26" Tinsley telescope was removed and the existing 20" telescope, manufactured by RC Optical Systems, was installed. Although the installation of the device was led by Nate Lust, students played a significant role in the effort. From the beginning, the telescope was a partnership between the University of Central Florida (UCF) and its students.

The telescope served faithfully from its installation until approximately three years ago, when its functionality began to degrade. A proprietary controller, manufactured by Bisque TCS, is at the root of the problem. This controller translates commands from the astronomy software package *TheSkyX*, which runs on a PC, into commands that drive the motors of the telescope through pulse width modulation (PWM) signals. The performance of this controller has deteriorated to the point where any stellar bodies tracked appear as blurs. There is a single individual that services this equipment, and he must be flown out at a significant expense for any repairs.

The original tasking for this project consisted of the design and integration of an open-source replacement for the existing Bisque TCS controller. However, due to the high cost of some of the related equipment (e.g., two Pittman motors valued at approximately $5,000 each), along with the uncertain nature of a project based exclusively around reverse engineering, the scope of the project has changed. Instead of replacing the controller at the Observatory, an interdisciplinary team (consisting of Computer Science, Mechanical Engineering and Electrical Engineering students) has been charged with designing an open-source scale model of the telescope.

II. Motivation and Goals

Although the scope of the original project has changed since its inception, this design will still serve two key purposes. First, the scale model with mitigate the risk of damaging costly observatory equipment as a permanent replacement controller is implemented. This scale model also serves as a test environment for subsequent Senior Design teams in later semesters. As the model has been designed to mirror the existing equipment in the Observatory, these teams will be able to quickly understand the current, as-is system. In addition, this model will serve as a low-risk test platform from which these teams can test and verify their own proposed designs.

The second intent of this design is for the hobbyist astronomer. These amateur astronomers need only provide their own telescope; all other parts will be readily available (e.g., 3D-print the mount, have the PCB fabricated from Gerber files and download the open-source astronomy software from the internet). With equivalent currently available solutions costing in the high hundreds-to thousands of dollars, this design will lower the bar of entry for an automated, computer-controlled telescope mount.

Finally, it bears mentioning that the intent of the Electrical Engineering team was for this design to be robust enough to serve as a replacement for the Bisque TCS controller, if the Observatory staff decided to implement it. Although the team does not anticipate that this option will be elected, the controller is capable of serving as a 1-to-1 replacement for the existing proprietary hardware.

III. System Layout

This following section briefly details the existing layout of the complete system at the Robinson Observatory, then narrows in focus to address the specific tasking and role of the Electrical Engineering team.

A. Existing System at Robinson Observatory

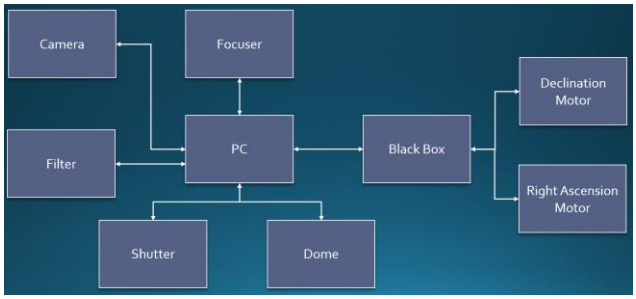

Fig. 1. A generalized block diagram of the existing as-is telescope system currently in place at the Robinson Observatory.

It is instructive to present the scope of the existing system at the Observatory to serve as a backdrop upon which the Electrical Engineering effort can be considered. This section is brief, as the true focus of the team is on the "Black Box" labeled in the block diagram above.

In general, the Observatory is controlled by a single computer (PC) that is integrated with various controllers across a wide range of functionality. Through various suites of software, the PC controls the rotation of the dome as well as the status (i.e., open or closed) of the shutter, along with a filter, a focuser and a camera, all of which are integrated with the telescope.

The area of concern for our team is what has been labeled here as the "Black Box." This is the proprietary Bisque TCS controller that is responsible for translating control signals from *TheSkyX* software suite into positioning control signals for the motors in the telescope mount. Specifically, two motors control the telescope: right ascension, which can roughly be considered a longitude position, and declination, that is: a position in the sky, typically referenced from -90 degrees to 90 degrees.

B. Design of the Motor Controller

As outlined above, the efforts of our team have been restricted to the "Black Box," or motor controller. The motor controller is comprised of a set of constituent parts, to include: a microcontroller, a joystick, optical sensors, indicator lights, a USB connection to the PC, a power supply, and motors and their associated encoders. Fig. 2 below summarizes the general block diagram for this controller, with additional details on each component to follow.

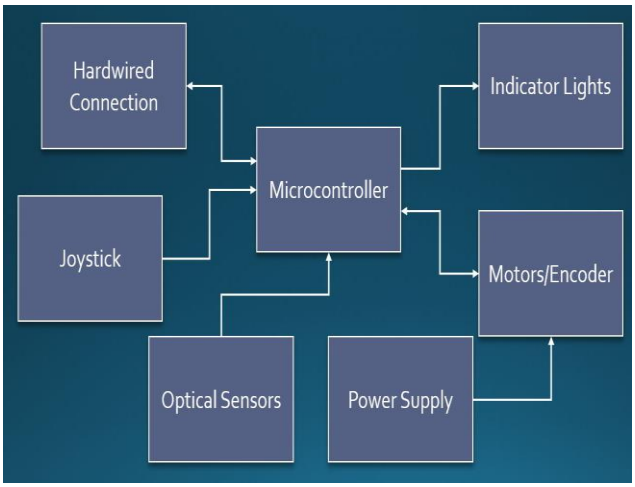

Fig. 2. A more refined block diagram of the "Black Box" of Fig.1. This represents the design of the Electrical Engineering team.

C. Microcontroller

It is important to mention, before proceeding with a discussion of design, that our PCB is generally a "shield" that mounts on top of the Arduino Mega 2560 board. This is per the request of our sponsor, the Florida Space Institute (FSI), and the intent is to facilitate ease of field replacement. If the Arduino Mega 2560 ever fails, it should be a relatively easy replacement for the end user.

Our design encompasses a pair of microcontrollers, both of which use the Arduino architecture. The ATmega2560 sits at the heart of the design, with the ATmega328 serving an ancillary purpose. The ATmega328 has had some of the LED load shifted to it, but its primary purpose is to serve as a "heartbeat" monitor – that is, if communication between the primary ATmega2560 and the shield is disrupted, the ATmega328 will notify the operator through the use of a trouble LED.

In general, much of the functionality of the ATmega2560 is used in this design. The ATmega2560 provides a 10-bit analog to digital converter (ADC), which can enable any one of the 16 analog pins after selection from a MUX that sits in front of the ADC [1]. This functionality is used for the only analog input in our design – the joystick. More detail on the specific implementation is included in the joystick section below.

The ATmega2560 natively supports 6 digital interrupt pins (2, 3, 18, 19, 20, and 21). All six were implemented in this design, with two pins each corresponding to the differential A and B signals from the motor encoders

(routed through a differential line receiver on the PCB) and two pins allotted to the optical switches that serve as limit switches and home position sensors.

The motors are driven by pulse frequency modulation (PFM), which is distinct from the more customary pulse width modulation (PWM) that may be expected. The duty cycle remains fixed, but the pulse frequency is varied. The Arduino 2560 only natively supports pulse frequencies of up to 490Hz; therefore, we sought an alternative. Here, we use a pre-defined Arduino library [2] that supports a range of 1Hz to 2MHz (on a 16-bit timer) and 31Hz to 2MHz (on an 8-bit timer). The equation used for calculating the necessary pulse frequency is detailed in the motors section of this paper, but this range is more than adequate for our implementation.

The hardwired connection to the PC is accomplished through the use of the Universal Serial Bus (USB) protocol. The Arduino Mega 2560 has a built in USB/Serial converter. This means that it will natively convert the incoming traffic from the PC (USB) to the 5V TX and RX signals that the Arduino expects (as well as converting the other way, to facilitate Arduino to PC communication).

Finally, the ATmega2560 controls a number of indicator LEDs. These LEDs are tied directly to the 5V digital I/O pins on the microcontroller. A resistor is placed in series to limit current and achieve the proper voltage drop across the LED.

D. Motors and Encoders

The pair of motors implemented in this design are Applied Motion STM17R-3NE, NEMA 17 units. These motors support a number of different control options. Since the motor and mount will be housed less than a few feet away from the microcontroller, and a large amount of environmental noise was not anticipated, we elected not to implement a differential line driver in our design and utilized the configuration detailed in Fig. 3 below.

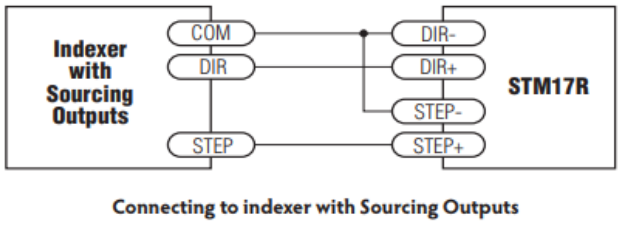

Fig. 3. The single step and direction connection from Arduino Mega 2560 digital I/O pins to STM17R-3NE motors [3].

Using the scheme outlined above, the step signal pulses once for each motor step and the direction signal is used to determine direction with a simple high/low input. The step size on this motor is dipswitch-configurable and can range between 200 counts per revolution (CPR) to 25,600 CPR. A simple equation (1) is used to derive the desired pulse frequency for the motors.

$$Pulse\ Frequency = RPS * Step\ Count \quad (1)$$

There are a number of other options available on these motors and will be dependent on the final design and requirements of the mechanical engineering team. Therefore, these options are addressed in the integration section of this paper.

We next consider the encoder. The STM17R-2NE supports a built-in quadrature incremental rotary encoder. The encoder follows the industry standard 26C31 differential line driver for output, meaning that the channels (A, B, and Index) are differential signals. Therefore, a differential line receiver (industry standard 26C32) is required to translate this output before it hits our microcontroller.

For position control, there are three primary signals of interest being sent by the encoder. Those signals are A, B, and Index. The A and B signals are the quadrature signals. That is, depending on the direction of rotation, one signal will lead, and the other signal will lag. The methodology for this is addressed in the software section of this paper.

The encoder bundled with the STM17R-3NE supports 1,000 lines of resolution. This means that for each rotation of the motor, 1000 "counts" are sent from the encoder. Our microcontroller tracks this encoder count, ticking upward or downward in increments of one for each clockwise or counterclockwise signal from the encoder. Therefore, as discussed in the microcontroller section of this paper, it is important for the A and B channels from each encoder to be tied to digital interrupt pins.

E. Power Supply

Much of the discussion on the power supply (PSU) is covered under the component selection portion of this document. The PSU is included here inasmuch as it pertains to the PCB design of the Arduino shield. Since the motors operate at a peak current draw of 2A each, we have elected to place 2, 2A fuses on our PBC to protect the motors.

F. Optical Switches

The operation of the optical switches is fairly straightforward. They serve a dual purpose, serving as both a limit switch and a home position sensor. This is

accomplished through the design of the interrupter wheel, as implemented by the mechanical engineering team. Although their final design was not available by the publication of this paper, a similar implementation from the Bisque TCS manual illuminates the application.

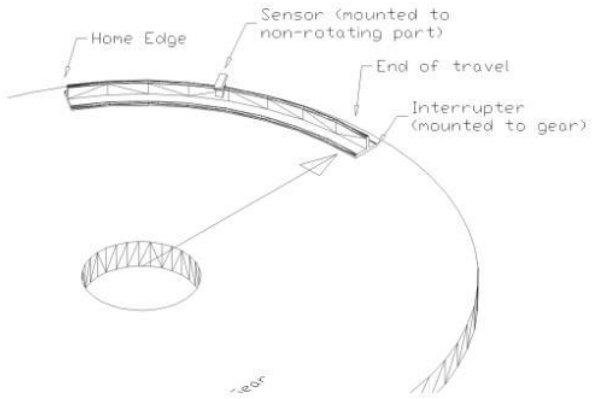

Fig. 4. The optical switch (sensor) can be interrupted by either reaching a limit or reaching the home position.

In general terms, the configuration of these OPB980 optical switches is simple. An input diode emits an infrared (IR) light which is intercepted by an optical sensor. This optical sensor determines whether the optical switch has been tripped or not.

We have elected to use the 3.3V output of the ATmega2560 to drive these optical sensors. A simple application of Ohm's law provides just over the necessary forward voltage (V_F) of 1.6V and expected current of 20mA with the use of an 85Ω resistor.

The output of these optical switches, at a minimum, is listed in the datasheet as $V_{CC} - 2.1V$. With a V_{CC} to the sensor of 5V, this provides a minimum of 2.9V. In accordance with the ATmega2560 datasheet, as well as extensive testing, this is more than sufficient to trigger a high input on the digital I/O pins.

G. Joystick

The joystick consists of three outputs. First, X and a Y-axis values are determined by a pair of 10kΩ potentiometers. These potentiometers vary the voltage out between 0 and 5V. These are the two connections that are routed to our analog pins on the microcontroller to make use of the ADC.

As a 10-bit ADC, we can extract 1024 levels of differentiation between 0 and 5V. However, as with many mechanical devices, it seemed prudent to program in a "dead zone" to prevent unintended joystick inputs from the center range of the ADC. Although it is a simple matter to modify in the code, our current dead zone exists from levels 500 – 530.

The third output on the joystick is a simple digital "button," which is either high or low. When this button is pressed, the user can take over manual control of the motors and direct them via joystick inputs. When the button is pressed a second time, manual control is released and the right-ascension motors return to tracking.

IV. Component Selection

This section revisits the major components of the design and addresses some of the considerations that went into their selection. In the interest of full disclosure, there will be a common theme throughout this section.

It was the intent of the Electrical Engineering team to mirror the existing equipment in the Robinson Observatory as closely as possible. As outlined in the introduction to this paper, this served a variety of purposes. The most significant considerations were the potential for this device to serve as a 1-to-1 replacement for the Bisque TCS controller, as well as to provide as much fidelity as possible for senior design teams who follow behind us and wish to use this as a scale model and starting point for their own development work. As such, although a variety of options and design parameters were considered during part selection, our main litmus test was: does this match (in form and function) what is currently in this Observatory?

A second concern is also common to the selection of each of these parts. That is, availability. For each component, it was essential that we made selections that were available from a variety of suppliers (e.g., Digikey and Mouser, at a minimum) with a large in-stock availability (typically over 1000 parts) and no apparent end-of-life or obsolescence on the horizon. Since this consideration was common to all parts, it is only mentioned here in the introduction.

A. Microcontroller

With the above preamble set aside, it is fair to say that the microcontroller was one of the parts least affected by the need to mirror existing observatory equipment. With that said, however, we had a specific request from our project sponsor, the Florida Space Institute (FSI).

Per FSI's request, we have elected to implement an Arduino shield. The general reasoning with the shield is that the core of the controller (that is, the Arduino development board) is easily replaced by an end user. Although we did investigate other options, such as the Raspberry Pi, they did not meet the request of our sponsor and were otherwise unsuited for our project. For example, we had no need for an operating system and were far more

comfortable with C than Python, which meant that the Pi would have been a poor design choice.

The selection of the ATmega328 was fairly simple once we had settled on the Arduino development environment. The general purpose of the ATmega328 is to serve as connection monitor between the shield and the Arduino Mega 2560, illuminating a trouble LED if a connection is lost. Moreover, we were able to offload some of the LED load to the ATmega328. The third, and perhaps most significant reason for adding the ATmega328 is so that design meets the requirements of the program for substantial PCB design. In truth, this design could have been accomplished without the ATmega328 – but its presence adds complexity to the PCB that is needed to meet design requirements.

B. Motors and Encoders

In general, the selection of the motors used on this project was driven by two criteria. First, as discussed above, it was our desire to closely match the existing equipment in the observatory. Since DC stepper motors were in use in the observatory, this drove our design decision as to the overall type of motor.

The secondary consideration for the motors used in this project was robustness. That is, with the design of the mechanical engineering team in flux until near the end of this project, it was necessary for our motors to support a wide variety of potential design choices.

At 68 oz-in of holding torque, our peers on the mechanical engineering team have assured us that these motors will meet any design specifications that they implement. However, the motors also support a number of dipswitch-configurable options, to include: idle current, pulse-noise filtering, smoothing filter, counts per revolution (CPR) and load inertia. These configuration options are examined in further detail in the integration section of this document but are included here to underscore the selection of a flexible motor.

Other general considerations included the very high CPR that can be achieved through this motor (ranging as high as 25,600 CPR). This enabled maximum flexibility to work with the mechanical engineering team, as their gear ratios (as it stands presently) range from 100:1 (for the declination motor) to 38,100:1 (for the right-ascension). The flexibility in the range of CPR allows us to use pulse frequencies within a reasonably tight range, rather than ones that differ by several orders of magnitude.

The election of a motor with a built-in encoder was, as with many other design choices, primarily driven by a desire to match the existing equipment in the observatory. As the existing Pittman motors use a built-in encoder, our technology was selected to closely mirror the as-is system. Moreover, the built-in encoder made the overall integration of the motor/encoder system more straightforward. Since this combination has been thoroughly vetted by the manufacturer, it (very likely) eliminated one potential point of failure and allowed us to concentrate troubleshooting efforts on other areas. Finally, the 1,000 line encoder provided sufficient resolution for our application, especially when considered with gear ratios ranging upwards of 38,000:1.

C. Power Supply

The primary consideration when selecting this power supply was to enable future extensibility. The current set of motors at the Observatory draw significantly more power than our scale model, but investing in a robust power supply at the onset of the project allows future teams to broaden the scope of the design without the necessity of replacing this major component.

The power supply selected for this project was recommended by the manufacturer of the motors, Applied Motion. It is an Applied Motion PS150A24 (150W, 24V DC motor). A secondary consideration regarding this power supply was the potential for regeneration if a regulated power supply was used. In brief, regeneration occurs when a load is rapidly decelerated from a high speed and much of the kinetic energy of the load is transferred back to the power supply. This regeneration can trip the overvoltage protection of a switching power supply, causing it to shutdown [3]. The vendor does offer a "regeneration clamp" to remedy this potential issue, but with the added cost of the additional protection, the manufacturer recommendation was all the more appealing.

D. Optical Switches

The optical switches selected for this design are a pair of TT Electronics OPB980T51Z switches. The major concern here was availability of the part in conjunction with a desire to match the existing equipment in the Observatory. Design decisions included flying leads (to ease integration with the PCB) and covered apertures (to enhance robustness).

E. Joystick

The design considerations for our joystick were functionality, cost and flexibility. The joystick that we selected met the functionality criteria better than others that were evaluated: that is, in addition to the X and Y-axis functionality of a standard joystick, it also included a push-button toggle. This push button was essential for our

implementation of the project (e.g., the ability to toggle in and out of manual control/tracking modes).

Moreover, this joystick, along with being the lowest cost joystick that we considered, also included a breakout board. This gave us the flexibility to incorporate the joystick into our PCB, if desired, or to utilize it as a stand-alone device without the need for additional PCB design. Ultimately, due to the design of the mechanical team, this joystick was implemented as a stand-alone component, and therefore, the breakout board is in use.

V. Software Design

The general approach to the code for this project follows the Arduino standard. That is, a setup routine executes, followed by the main loop(). In our case, the setup routine initializes timers (per the PWM library), sets pin modes (input, output, input_pullup) as needed, initializes serial communication and executes a homing function.

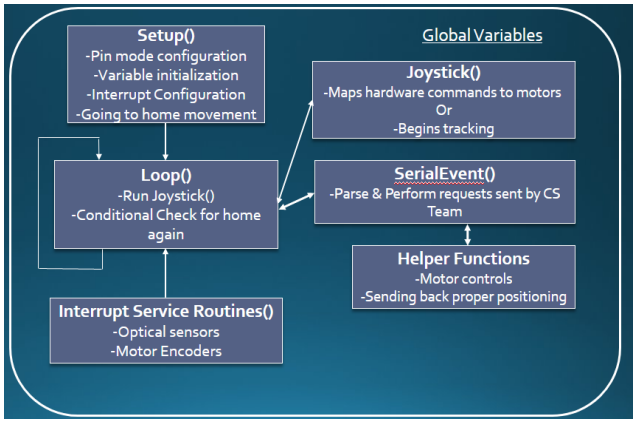

Fig. 5. A general block diagram detailing the overall software implementation.

The homing function turns right-ascension (RA) and declination (DEC) motors counterclockwise until the optical switch is tripped. When both optical switches are engaged, the telescope is homed. Encoder counts are reset to zero, which translates to a 0° RA position and a -90° DEC position.

In broad terms, the main loop is always polling for serial input data. If there is serial data available from the PC, the software parses the command. There are three possible options for these commands. The first two options are a request for our motor position. The position is translated into an HH:MM:SS format for RA and DEG:MM:SS format for DEC and returned to the PC via serial write. The third option is a motor command. In the case of a motor command, the software compares the requested position to the current position and continues to pulse the motors until the two are even (within a tolerance, which was required to prevent motor jitter). Once the position is reached, a tracking function executes, setting the motors to the correct speed to compensate for the earth's rotation.

In addition, the software is also looking for a button press from the joystick. If the button is pressed, control is passed to the joystick. The joystick can be used to manually position the telescope mount. When the button is pressed a second time, the tracking function executes.

There are several techniques that were used in the software that are worth an individual mention. In each case, these are not especially advanced techniques, but are essential to the successful operation of the program.

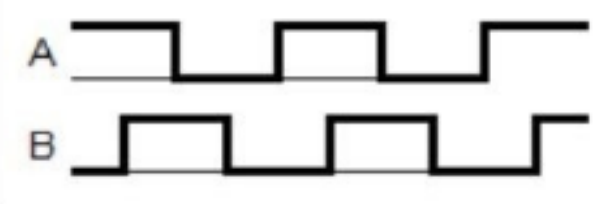

Fig. 6. A timing diagram demonstrating a hypothetical clockwise (CW) rotation. Channel A (rising edge) triggers the interrupt, channel B is low. This suggests CW rotation.

First, we consider the handling of the interrupt service routines (ISRs) for the encoders. To correctly determine the rotation direction (clockwise or counterclockwise) of the motors, we need to understand whether channel A or channel B on the encoder is leading the other channel. To do this, we execute the interrupt on the leading edge of channel A. We then check the state of channel B. If channel B is low (as referenced in Fig. 6), we interpret clockwise rotation (channel A leads). If channel B would be high, we interpret CCW rotation (channel A lags). A secondary pair of ISRs are attached to the optical switches, as the limit switch must take priority over any other code execution to prevent possible damage to the telescope.

We next consider the joystick button itself. Throughout several implementations of our code, and both revisions of the PCB, we found that the push-button had intermittent (at best) operation. Research into the issue revealed that the software needed a simple debounce feature. This is accomplished quite directly, by simply checking the state of the button, inserting a brief delay, and then confirming that the button has changed state.

The final consideration of the code relates to the variables used in the ISRs. In the early revisions of our code, we found that the encoder would seemingly stop counting (that is, incrementing and decrementing). Since the encoders were tied to ISRs, this seemed like an unreasonable conclusion. The ultimate fix was to declare the shared variables as volatile. This tells the compiler that such

variables might change at any time, and thus the compiler must reload the variable whenever you reference it, rather than relying on a copy it might have in a processor register [4].

VI. Integration

Working alongside mechanical engineering and computer science students as part of an interdisciplinary team requires significant efforts towards integration. Therefore, this section will address the integration efforts with each team separately. In the interest of full disclosure, the mechanical engineering prototype will not be complete until after the time of the senior design presentation; therefore, mechanical engineering integration details will consider theoretical options.

A. Computer Science

The integration work with the computer science team began in earnest around the middle of the Fall semester. The goal here was for interoperability between their custom Stellarium software and our motor controller. Simply put, the end user needed to be able to select a stellar body (or set of coordinates) in the Stellarium software and our controllers needed to point the telescope at it and begin tracking.

This necessitated the three different commands discussed in the software portion of this paper. Two commands can be issued from Stellarium to request the position of the mount. That is :GR# (to request the RA position of the telescope) and :GD# (to request the DEC position of the telescope). Our motor position is tracked internally based on the encoder count, which is cross-referenced against the gear ratio to determine how far the mount has moved. This is then translated by a pair of functions into the expected Stellarium format (e.g., HH:MM:SS or DEC:MM:SS).

Once Stellarium has determined the position of the mount, it can then calculate the new position request. It does this by sending a string, deliminated by the "-" character (e.g., M-120.2-50.5). This string commands the motors to move, then references an absolute degree from our home position.

Through a constant series of position requests, Stellarium is able to track the movement of the telescope across the sky as it slews towards its intended target. Similarly, the constant position requests allow the software to accurately reflect the ability of the controllers to track against the earth's rotation. These three features combine to allow the user to command the telescope to any position in the sky, while simultaneously being able to visually verify the position of the telescope against the Stellarium software.

B. Mechanical Engineering

Although integration with the mechanical engineering team has not been completed as of the date of this document, we can nevertheless consider some of the possible adjustments that will be made to accommodate their design. Some of these accommodations are made through our software and other accommodations are enabled by our selection of motors.

First, we consider the software accommodations. The gear ratio is a simple declaration in our code and allows for a range of possibilities. At the time of this writing, the gear ratios will be 38,100:1 (for right-ascension) and 100:1 (for declination). These gear ratios, combined with our step count (counts per revolution), inform the pulse frequency required to drive the motors at the desired speed for slew and tracking (per equation (1)).

We also have a number of hardware configurations that are afforded to us by the motor selection. Since we are unsure of the mechanical requirements of the final mount design, flexibility here allows for fine tuning of areas such as heat and power. All options are configured via dipswitch on the motor.

We first have the ability to tune both current and idle current. Current affects maximum torque, while idle current affects holding torque. Current can vary between 50% and 100% of maximum. The dividends in reduced heat are not linear to the reduced maximum torque, however. For example, at 70% current, the motor produces 70% of the rated torque but only 50% as much heat. Therefore, fine tuning these variables will lead to increased power efficiency and expected lifespan of the hardware.

A smoothing filter is also offered and can simply be configured as "off" or "on." The smoothing filter is recommended when the motors are not micro-stepped (2000 steps/rev and beyond). This allows for smooth motion from coarse command signals. The drawback is a slight delay, or "lag" in the motion. In our testing, this delay has been negligible, but we will need to complete full integration with the final mount design to know if this option should remain enabled.

The final consideration is a step pulse noise filter. This is a digital noise filter included with the STM17R that helps to overcome electrical noise. This electrical noise may otherwise cause the drive to interpret a single step pulse as two or more pulses. Configuration options here depend on the frequency that will be sent to the drive. The breakpoint here is above or below 150KHz. We expect to operate well below 150KHz, but this is another area that will not be set in stone until after the mechanical engineering integration.

VII. Conclusion

The objective of this design effort was twofold. First, we set out to design a scale model with which subsequent senior design teams could further advance repair efforts for the Robinson Observatory. Second, we desire to create an open-source platform for hobbyist astronomers to be able to more readily pursue their hobby. We can say with confidence that these goals, from an electrical engineering perspective, have been accomplished. The prototype communicates with the Stellarium software and performs as expected. Integration with the mechanical engineering team is pending as of the date of this submission, so the outcome there is less certain – but it is certainly fair to say that progress has been made towards both goals that we set forth to accomplish.

Acknowledgement

The authors wish to acknowledge the assistance and support of the Florida Space Institute (FSI)/Florida Space Grant Consortium (FSGC) as well as Dr. Yan Fernandez and Dr. Zoe Landsman of the University of Central Florida (UCF) Department of Physics.

References

[1] B. Thomsen, "Analogue input," *Embedded Engineering Design.* [Online] [Accessed November 6, 2019].
[2] runnerup, "PWM frequency library," *Arduino Forum.*, [Online] [Accessed November 6, 2019].
[3] "Hardware manual: STM17R drive+motor", *Applied Motion*. [Online] [Accessed November 7, 2019].
[4] N. Gammon, "Interrupts," *gammon.com.au*. [Online] [Accessed November 7, 2019]

Anthony J. Eubanks, Jr. is a senior Electrical Engineering student at the University of Central Florida. He has received a previous degree in Electrical Engineering Technology from Western Piedmont in Morganton, North Carolina.

He is currently a member of the Science Mathematics and Research for Transformation (SMART) program through the Department of Defense (DoD). Upon graduation, he will be working at the U.S. Army Space and Missile Defense Command (SMDC) in Redstone Arsenal, developing radar and high-energy laser systems.

Brian T. Glass is a senior Electrical Engineering student at the University of Central Florida. Prior to attending UCF, Brian spent 10 years working for Guardian Protection Services in Pennsylvania.

He has spent two summer internships at Northrop Grumman, both in Baltimore and Orlando. Upon graduation, he has committed to working at Lockheed Martin Missiles and Fire Controls in Orlando, Florida.

Melinda I. Ramos will be graduating with a bachelor's degree in electrical Engineering and a minor in Intelligent Robotic Systems. She is currently working for Lockheed Martin MFC as an Advanced Manufacturing Technologies Intern. After graduation, she is transitioning to a full-time position in the Test Engineering department and hopes to join the Engineering Leadership Development Program class of 2023 at Lockheed.

Thomas A. Vilan is a senior Electrical Engineering student at the University of Central Florida. He is presently working at Bogen Communications. He will receive a minor in Computer Science, and his major interests lie in computer communications and MEMS fabrication.