A HEATR for the SHARP Polarimeter at the Caltech Submillimeter Observatory

Mitchell Drew

1. Cosmic Magnetic Fields and Their Role in Star Formation

For millennia, humanity has striven to find answers to the most fundamental questions of our origin, not only the origin of our species but also of Earth itself and of every living organism that inhabits it. Questions like "Where did we come from?" and "How did we get here?" still puzzle the scientific community, and many of the pieces have yet to be found. However, recent technological advancements in physics and astronomy have led to numerous exciting discoveries that have brought us a few steps closer to completing the picture of our existence. For example, twenty years ago researchers had not yet confirmed the existence of a single extrasolar planet, a planet orbiting a star other than our own Sun, but today NASA's PlanetQuest database boasts nearly 700 confirmed extrasolar planets and more than 2,000 candidate planets still awaiting confirmation (2). NASA's Kepler mission, designed to survey over 100,000 stars in its search for Earth-like planets, has observed all of these candidates and has confirmed sixty-one of them since its launch in 2009 (3).

Perhaps an even more significant question than the existence and number of planets similar to Earth is the problem of how these planets, and their parent stars, form. Astrophysicists do not yet understand all of the processes involved in star formation, but new instruments and methods of observation have allowed us to develop and test theoretical models to build a more comprehensive portrait of these processes. What we do know is that stars form within Giant Molecular Clouds, large regions of diffuse matter between the stars (4). These clouds can span hundreds of light-years in diameter and consist mostly of molecular hydrogen, although they also contain small concentrations of other particles, including gases such as carbon monoxide as well as microscopic clumps of solid matter (4). The density of particles in GMCs is not uniform; in fact, stars tend to form in regions called "dense cores," isolated regions that collapse under their own gravity as a result of the higher concentration of matter present within these regions (4).

Researchers have investigated the properties of star formation through numerous simulations and comparison of their results with the observations recorded of star-forming regions (4). Many agree that dense cores tend to collapse from the inside out: the material in the center clumps together initially, the gravity of this material begins to attract more matter from its immediate surroundings, and the increased mass strengthens the body's gravitational field, causing it to pull even more matter toward itself (4). The body in the center of this collapsing dense core is referred to as a protostar and is typically more than one million times smaller than the initial dense core region (4). Eventually, a circumstellar disk of rotating dust and gas forms around the protostar, and the matter in this disk is believed to clump together to form planets, asteroids, and other bodies that will revolve around the star (4). Before this, however, the protostar accumulates a few tenths the mass of the Sun and begins the nuclear fusion of deuterium, an isotope of hydrogen that requires significantly less energy to initiate fusion (4). Around this time, strong solar winds at the protostellar surface disperse much of the gas and dust in the dense core surrounding it (4). The source of these winds is unknown, as they are not a result of theoretical predictions; rather, they were interpreted from direct observation of gas moving outward from sources appearing to be protostars (4). One possible explanation for such observations is magnetic fields, which play a significant role in solar wind activity in our own solar system (5). Magnetic fields, it turns out, may play a vital role in star formation (6).

Throughout the galaxy, magnetic fields influence a variety of cosmic activity, including solar flares and the Aurora Borealis (5). Not only do these fields exist within stars and planets, but they also pervade the galaxy itself and affect the formation of interstellar gas clouds (5). From the observation of charged particles moving through the interstellar medium, the Milky Way galaxy appears to have a magnetic field on the order of 10^{-6} gauss, roughly 250,000 times weaker than the strength of Earth's magnetic field at its surface (5). The origin of these magnetic fields is not yet understood. (5). One of the most significant mysteries surrounding magnetic fields is their role in star formation. Models and observations of developing stars contain problems such as the conservation of angular momentum as a protostar accretes mass (6). One solution proposes that the

entanglement of the protostar's field lines, which carry a form of energy analogous to the tension in a stretched rubber band, causes the rotation of the protostar to slow down, allowing it to accumulate more matter (6). A competing theory states that the accumulation of matter can be attributed to a process called supersonic turbulent flows within the gas (7). The study of cosmic magnetic fields is fairly new because the methods of detection were developed so recently, so the numerous questions that researchers have regarding them can be found only through continued and improved observation. In particular, as described in the following section, submillimeter polarimetry offers a new method for mapping magnetic fields and is developing rapidly.

2. Submillimeter Polarimetry with SHARP and the SHARC-II at the CSO

In their early stages, protostars and the dense cores from which they evolve do not emit light at optical wavelengths (4). These early systems are surrounded not only by gas but also by dust grains that absorb the radiation emitted from the central protostar. The dust grains re-emit this radiation at longer wavelengths, and other grains absorb this radiation (4). The process continues until the wavelength of the light is long enough that the small dust grains can no longer absorb it, and it passes through them (4). By the time this radiation can be detected from Earth, it has been stretched to submillimeter wavelengths, nearly one millimeter, and cannot be observed using optical telescopes (4). Additionally, the dust grains emitting the light, although irregularly shaped, tend to be elongated, which results in the light that they emit having a linear polarization in a particular direction related to their spatial orientation (6). For reasons not yet fully understood, the elongated axis of a dust grain tends to align perpendicular to the ambient magnetic field (6). Therefore, using a new technique first utilized about thirty years ago called submillimeter dust emission polarimetry, astronomers have mapped magnetic fields in GMCs and star-forming regions from the polarized light emitted by the grains (6).

Figure 1: SHARP (stack of boxes at center) and the SHARC-II (cylindrical tower at right) at the CSO.

The first instruments used to measure polarized submillimeter dust emission each had only a single pixel for its detector (8). One of these submillimeter telescopes was a European balloon-borne telescope; the other was NASA's Kuiper Airborne Observatory, a submillimeter telescope flown aboard an airplane (8). Both projects needed to operate at as high an altitude as they could safely attain, for the water vapor in Earth's atmosphere absorbs far-infrared and submillimeter radiation very efficiently, and all subsequent submillimeter observatories have been either stratospheric (air- or balloon-borne) or ground-based at high altitudes in locations with little moisture in the air. One such location is the 14,000-foot summit of Mauna Kea on the island of Hawai'i (8). This peak contains many different observatories, one of which is the Caltech Submillimeter Observatory (CSO). At the CSO is the SHARC-II, the most advanced submillimeter detector as of 2007, as well as SHARP, a module developed at Northwestern that

converts the SHARC-II into a polarimeter (9). While many previous polarimeters had a wide field of view and could map large-scale regions of Giant Molecular Clouds, SHARP has a smaller field but much higher angular resolution (9), so it can be used to observe protostars and their immediate surroundings and map the magnetic fields within these small regions. Interpreting these maps in conjunction with those on a larger scale allows for the comparison of large area magnetic fields with those on the smaller scale of protostars and dense cores.

Another advantage of SHARP is its dual-polarization capability (9). The module contains a half-wave plate, which is a device that rotates incoming polarized light depending on its orientation (9). Polarizing grids then split the light into two orthogonal polarization components (9). While many applications of half-wave plates only utilize one of these components, SHARP's half-wave plate and mirrors are arranged to allow both states of polarization to be captured by the detector simultaneously (9). This property is very beneficial, as it provides twice the data that would normally be received as opposed to losing the information from one of the two polarization components. This feat is especially impressive when taking into account the spatial constraints placed upon the SHARP module, as explained below.

When in use, SHARP must fit between the telescope and the SHARC-II, both of which are fixed in place with only a few feet separating them (see Figure 1). As a result, some compromises had to be made in its design, one of which limited the space occupied by the width of the half-wave plate mount to a few centimeters. During observations, the half-wave plate is rotated to modulate the polarization angle. A rotary motor contained within the module controls this rotation. Typically, roller bearings would provide the most reliable mechanism for this application; however, the spatial constraints placed on the wave-plate bearing assembly necessitated a much thinner bearing. The design chosen was a needle-thrust bearing, a cylindrical bearing oriented to fit easily into tight spaces (10). Unfortunately the cylindrical bearings in the half-wave plate do not operate properly for this application. They perform well at room temperature but malfunction when the ambient temperature drops, which happens quite frequently at Mauna Kea's

14,000-foot summit. Currently, the part of the module containing the half-wave plate has resistors mounted to it that are attached to power sources, and the observer using SHARP must operate these power sources manually. As the temperature changes throughout the night, the observer must pay attention and adjust the voltage of the power supplies accordingly.

3. Honors Thesis Project: The SHARP HEATR Prototype

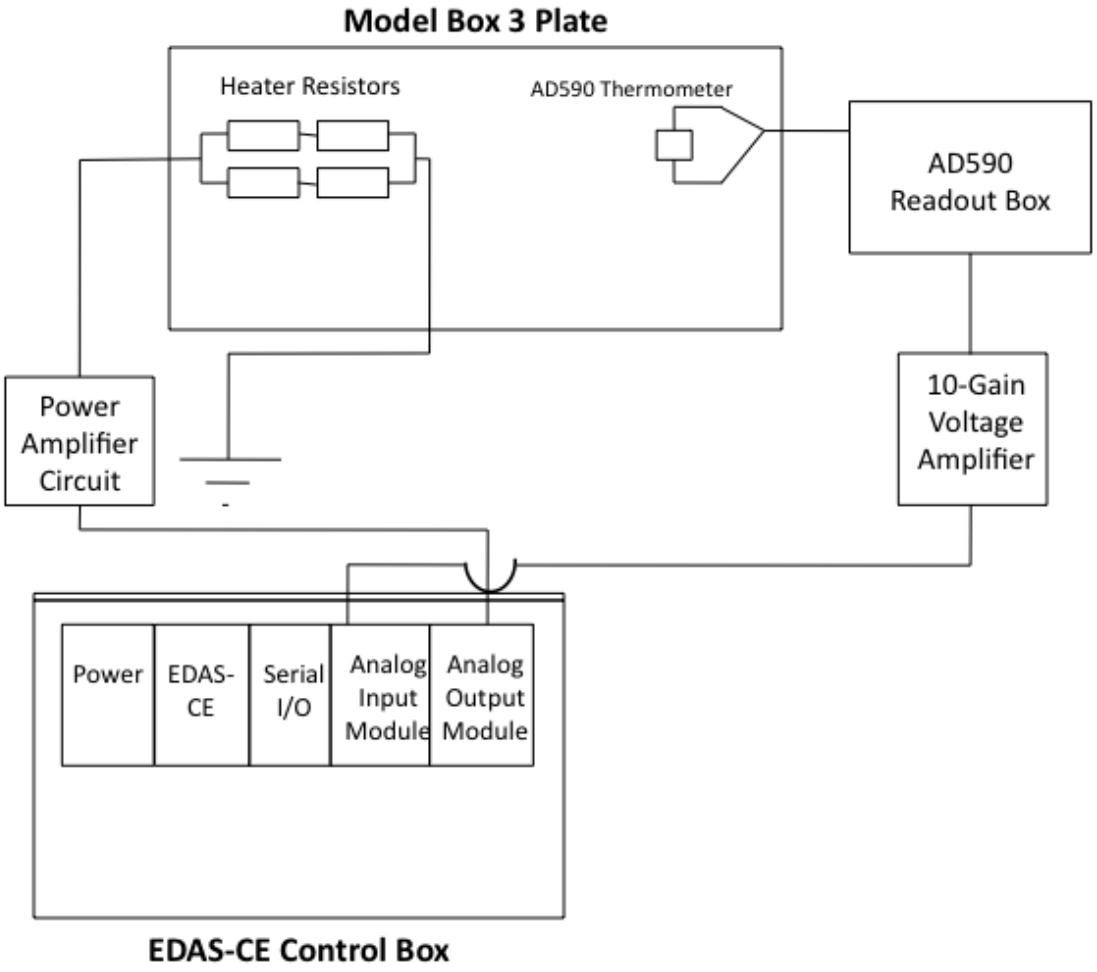


Figure 2: Schematic of the basic components of the HEATR system.

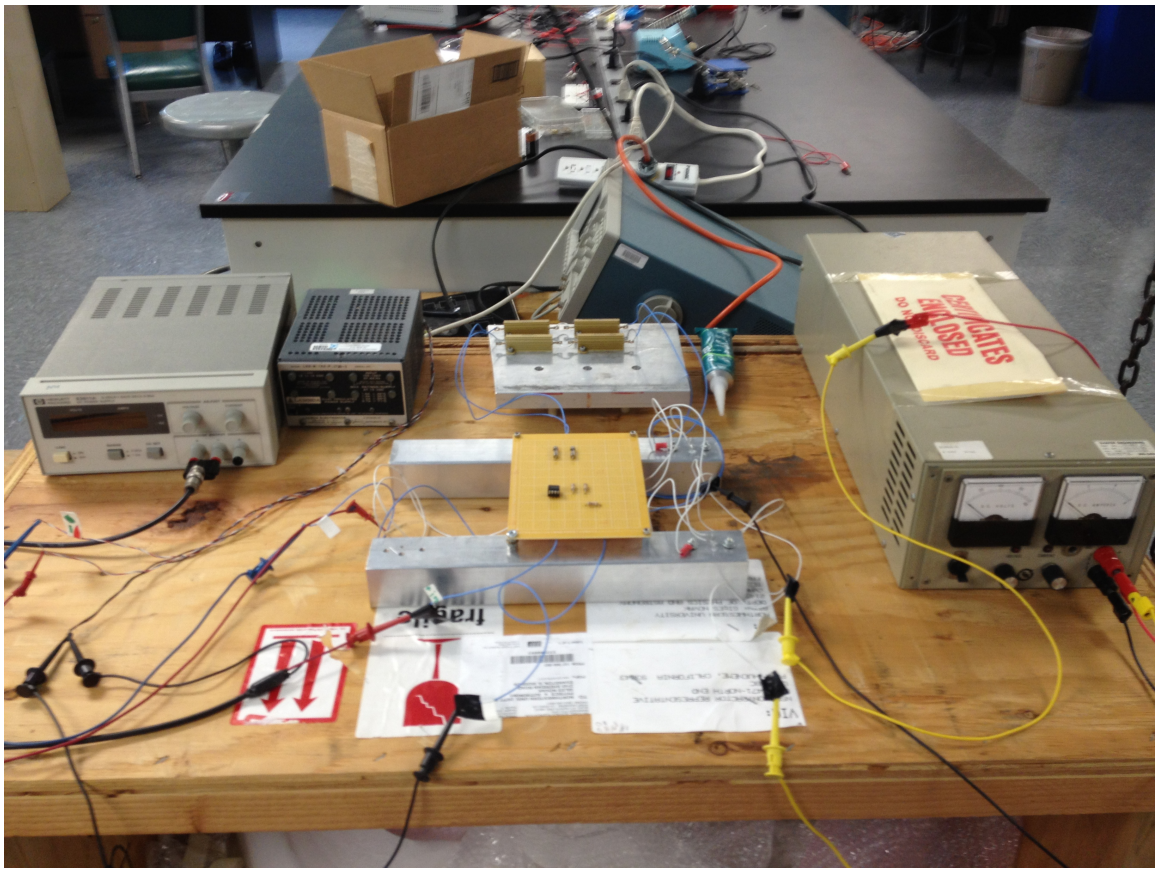


Figure 3: HEATR prototype power amplifier circuit and power supplies, with the resistors mounted to the test plate behind. Control computer and AD590 control circuit not shown.

My honors thesis project was to develop a prototype of an automated heating system for the SHARP wave-plate bearing assembly. We refer to this prototype as the Half-wave plate External Automated Temperature Regulator, or HEATR. It consists of a closed-loop feedback temperature control program, a series of circuits, and an aluminum plate designed to simulate the SHARP half-wave plate module. The program outputs a voltage from a computer, running it to a power amplifier circuit I built, which sends a high current to a series of resistors mounted onto a plate. These resistors heat the plate, and an electronic thermometer measures its temperature and sends this information back to the computer, which adjusts its output voltage accordingly. Incorporating HEATR into SHARP will save future data from being lost by eliminating human error associated with the manual adjustment of power supplies. It will also allow the observer to focus on other important tasks, as focusing and operating large machinery can be quite difficult at such a high altitude. Optimizing this project required matching the components in the

prototype to those to be used at the CSO. Most importantly, the aluminum plate to be heated needed to have the same thermal mass as the half-wave plate module in Box 3 of SHARP, to which the half-wave plate, motor, encoder, and heater resistors are mounted. Additionally, I needed to use the same resistors used in SHARP to heat my own plate to ensure that they dissipated power at the same rate. The heating system will be controlled by the EDAS-CE, an embedded computer currently at the CSO that interfaces with the motor controller of the half-wave plate. An identical EDAS computer system was used for my HEATR prototype. Since this embedded computer does not have a monitor or keyboard of its own, the program must be uploaded and executed through an external computer. Once uploaded, though, the program can be set to run automatically each time the EDAS computer is powered on, so manual execution will not be necessary at the beginning of every observing session.

As discussed earlier, the bearings that allow the half-wave plate to rotate operate optimally at room temperature. However, since the CSO sits at the top of the 14,000-foot summit of Mauna Kea, temperatures can drop well below freezing, especially at night during observing sessions with the dome opened. On top of this, the ambient temperature changes throughout the night, which currently requires observers to pay attention to temperature shifts and adjust the heater voltage supplies manually according to these changes. The HEATR therefore must be able to raise the temperature of the plate from roughly 20°F to about 75°F and hold it to within a few degrees of this temperature. Since observing time is precious, the heater certainly must bring the plate to the desired temperature within one hour of its activation (i.e. during twilight) and preferably should be able to reach its goal in half that time, approximately thirty minutes.

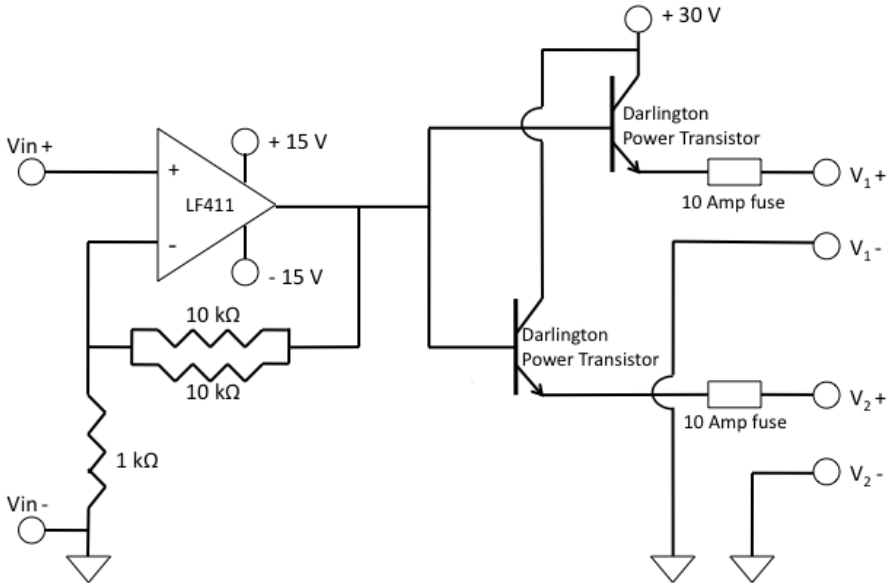


Figure 4: Power amplifier circuit. V_1 and V_2 each connect to a pair of heater resistors.

Since the aluminum plate used in the prototype has roughly the same thermal mass as the plate in Box 3 of SHARP, so tests of this system serve as a good approximation of how the HEATR will work at the CSO. The heater resistors, mounted and heat-sunk to the plate to maximize heat transfer, are each 5 ohms and have a power rating of 50 Watts. The resistors are connected in series in pairs, so each pair acts as one 10-ohm resistor and can dissipate up to 100 Watts. Supplying a voltage of 10 Volts across the resistor pairs, slightly less than is currently used during manual heating at CSO, requires a current of 1 Ampere, by Ohm's Law. While the EDAS computer's analog output module can supply 10 volts, it does not have the ability to output any more than 10 milliamps of current, so a power amplifier circuit must be used to provide the resistors with enough current to heat the plate sufficiently. A schematic of the circuit I built for this purpose can be found in Figure 4 above. Not only does this circuit provide a significant current gain, but it also has a voltage gain of 6, allowing a voltage drop greater than 10 volts across the resistor pairs. The gain in voltage comes from the circuit's input stage, a non-inverting amplifier made with an LF411 operational amplifier. The op-amp is connected to a supply voltage of +/- 15 volts, allowing it to output up to approximately 13.5 volts depending on the input voltage received. The next stage of the circuit, two Darlington power transistors,

provides the increased current via the 30-volt power supply connected to each transistor's collector. The base of each transistor receives the output voltage of the op-amp and, after a small voltage drop from base to emitter, passes this voltage to the emitter along with the current from the collector. Each pair of heater resistors is connected to one of the transistors' emitters with a 10-amp fuse to ensure that the circuit does not deliver too much power to the resistors. The transistors themselves are mounted and heat-sunk to the aluminum shafts that support the circuit and can dissipate power from the collector voltage supplies when they are not delivering it to the resistors.

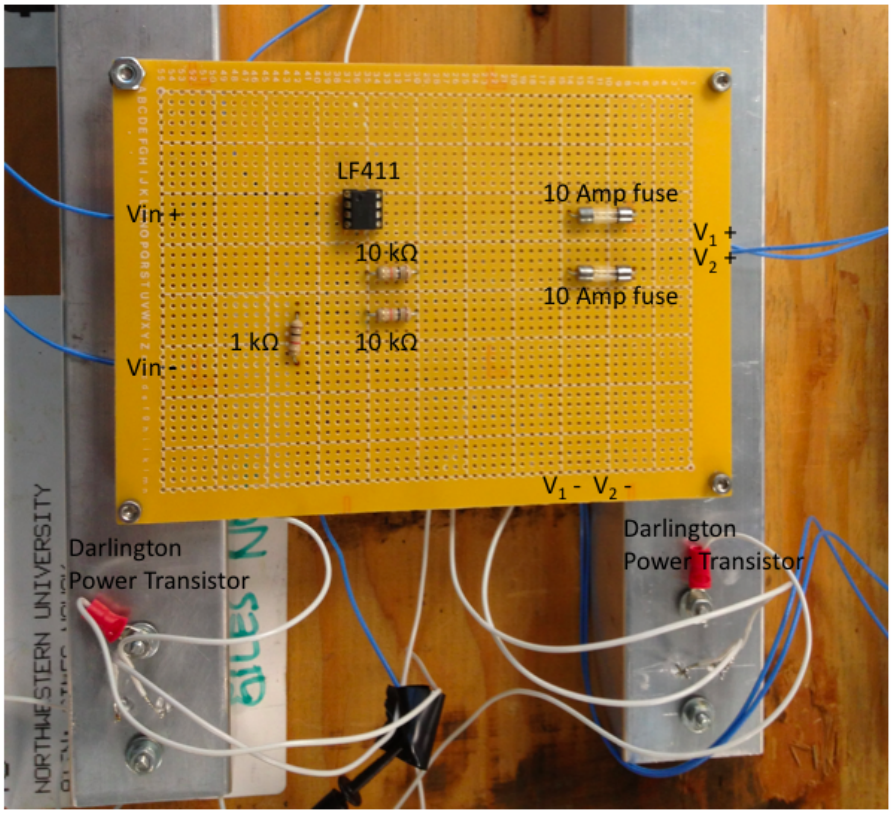


Figure 5: Power amplifier circuit, mounted to aluminum shafts.

In addition to the resistors responsible for heating it, the plate has an electronic thermometer, embedded in a thin sheet of aluminum, mounted to it with heat sink compound. This thermometer is an AD590 temperature transducer, an integrated circuit that outputs a current proportional to the absolute temperature that it measures, in Kelvin. The AD590 sends its signal to its control circuit, which in turn outputs a calibrated voltage (1 millivolt per Kelvin) to the analog input module of the EDAS. However, since this voltage output will never reach more than a few hundred millivolts, the signal first

passes through an amplifier with a gain of 10 to reduce the effect of noise from the analog input module as it reads the voltage. The EDAS then receives 10 millivolts for every degree Kelvin measured by the thermometer.

The EDAS-CE computer controls the entire heating system. Its analog input module reads the voltage from the thermometry circuit, and its analog output module sends a voltage to the heater circuit. Since it is an embedded system without its own display or keyboard input, programs intended to run on the EDAS must be compiled remotely and then uploaded. This can be done through a serial connection to the computer's serial input/output module, or it may be done through Ethernet. For this application an Ethernet connection makes more sense, since it provides a faster and more reliable line of communication. Additionally, connecting to the device via serial port requires close proximity to the device, whereas an Ethernet connection allows the user to communicate with the EDAS from remote locations, as long as he or she knows its IP address. This property is particularly desirable because the user may access the EDAS at the CSO directly through the Internet at Northwestern to run, debug, and recompile a program, rather than having to establish a connection with an intermediate computer at the CSO that communicates with the EDAS via serial port.

The EDAS-CE is not a widely utilized system, and we found that it has certain peculiarities in its operation that are not trivial to solve. Most notably, we discovered that the analog input module carries with it a voltage offset of about 1 volt, depending on the level of input provided. For an input voltage near ground, the offset is approximately 1.4 volts, while an input close to 10 volts, the maximum voltage capable of being read by the EDAS, receives an offset closer to 0.9 volts. Within the operating input voltage range of the HEATR, 2.5-3.2 volts depending on the ambient temperature and the desired temperature for the plate, the voltage offset remains nearly constant at approximately 1.2 volts. However, the offset was not always so predictable. Initially the voltage readings were about 2 volts too high when the offset was discovered. Then it began to vary, at times reaching as high as 6 volts above the voltage provided. Further, the offset would occasionally drift over this 2-to-6 volt range on short timescales, making it impossible to

get a temperature reading with error within one degree since one degree corresponds to 10 millivolts at the EDAS input. The most troublesome aspect of this issue is that it would actually induce an offset in the device supplying the voltage, most readily apparent on the display of the variable 20-volt DC power supply used during testing. After attempting various solutions, we concluded that the problem was likely a result of inconsistent grounding of the various devices involved in the system. I shorted the grounds of the analog input and analog output channels together and connected both of these to the case of the EDAS Control Box, which was grounded to the third prong of a nearby electrical outlet, and the offset improved immediately. In addition, the battery-powered amplifier used in the thermometer signal has a switch that allows it to operate in either grounded or differential mode. The grounded mode utilizes the input ground as its reference, and the differential mode isolates input from output. Switching the amplifier to differential mode eliminated the large-scale voltage drifts received by the analog input module. While the offset of 1.2 volts in operating range is still present, its consistency allows for reliable use of the EDAS in the temperature regulation system.

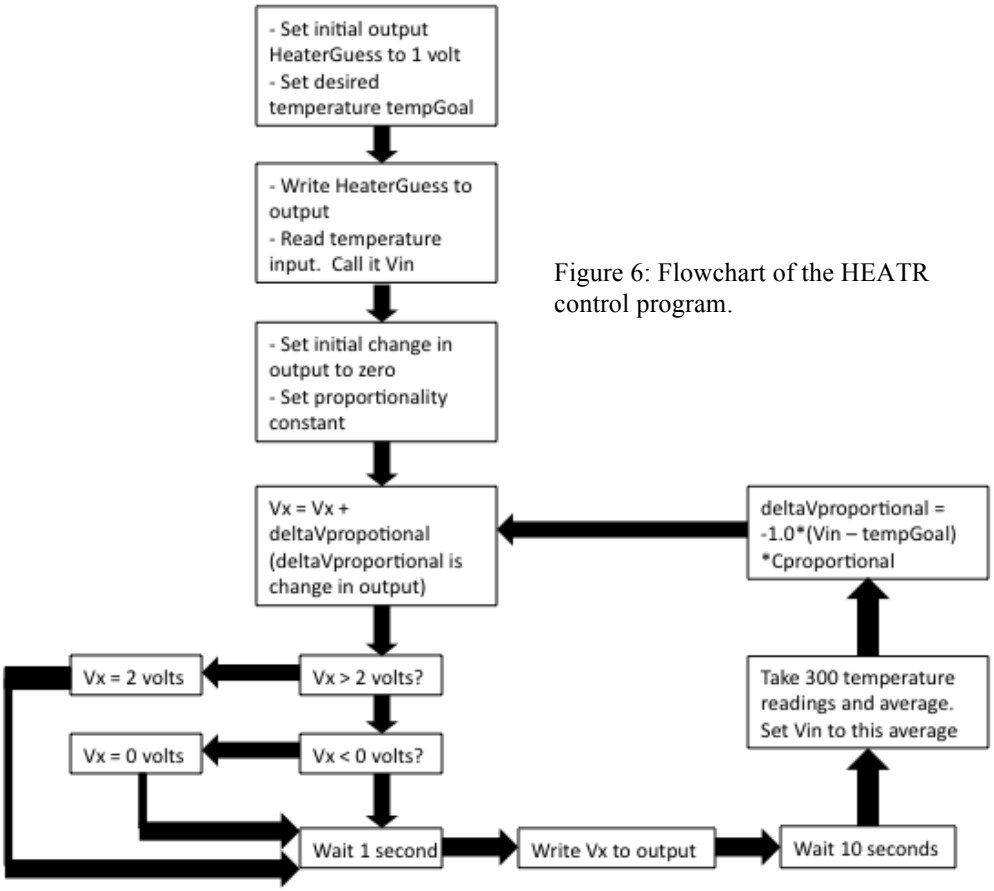


Figure 6: Flowchart of the HEATR control program.

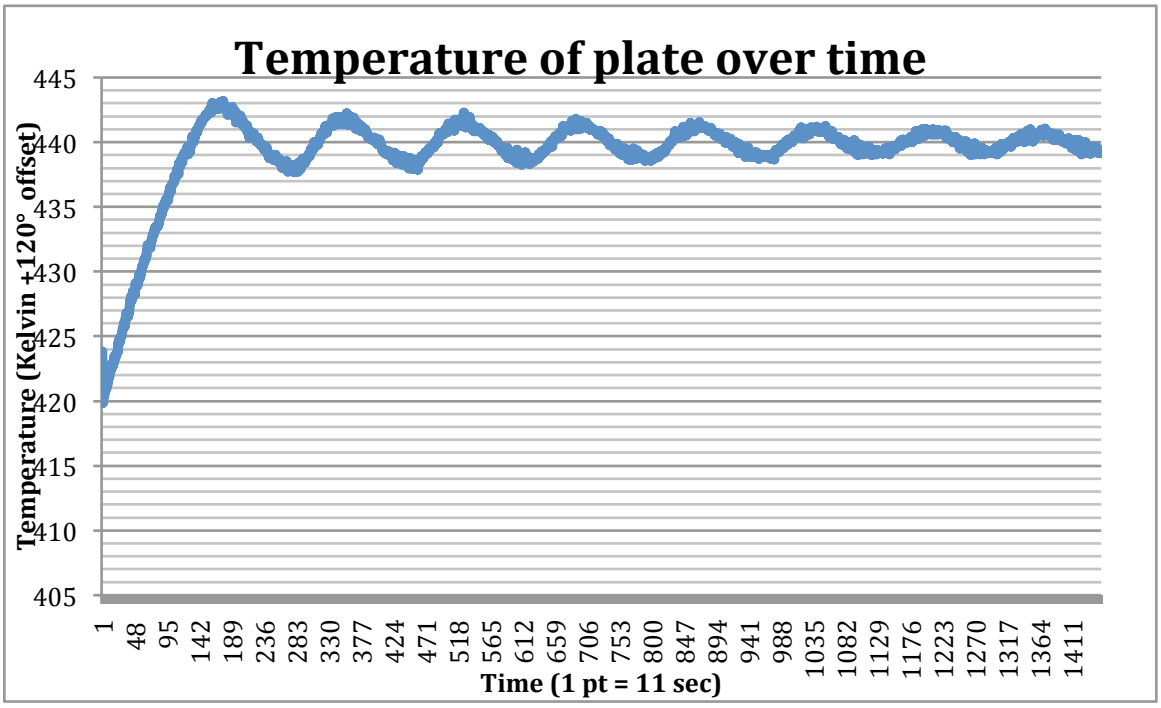
The program on the EDAS-CE that controls the heater is a basic closed-loop control system that outputs a voltage based on the difference between the desired temperature of the plate and the temperature currently being read by the AD590 thermometer. This code has been included in Appendix A for reference. Figure 6 above outlines the temperature control loop in the program. Many of the declarations and functions necessary to operate the EDAS are specific to the EDAS-CE software and are not used for any other applications. Fortunately the manufacturer, Intelligent Instrumentation, included simple test programs for each EDAS module on their website, so I managed to find one for the analog input module and another for the analog output module. After testing these programs individually to ensure that they would in fact compile and run on the embedded computer, I combined and modified them to verify that the system could read a voltage at its input and output a corresponding voltage. I then implemented an algorithm to read an input voltage from the thermometry circuit and adjust the output voltage accordingly in a continuous feedback loop.

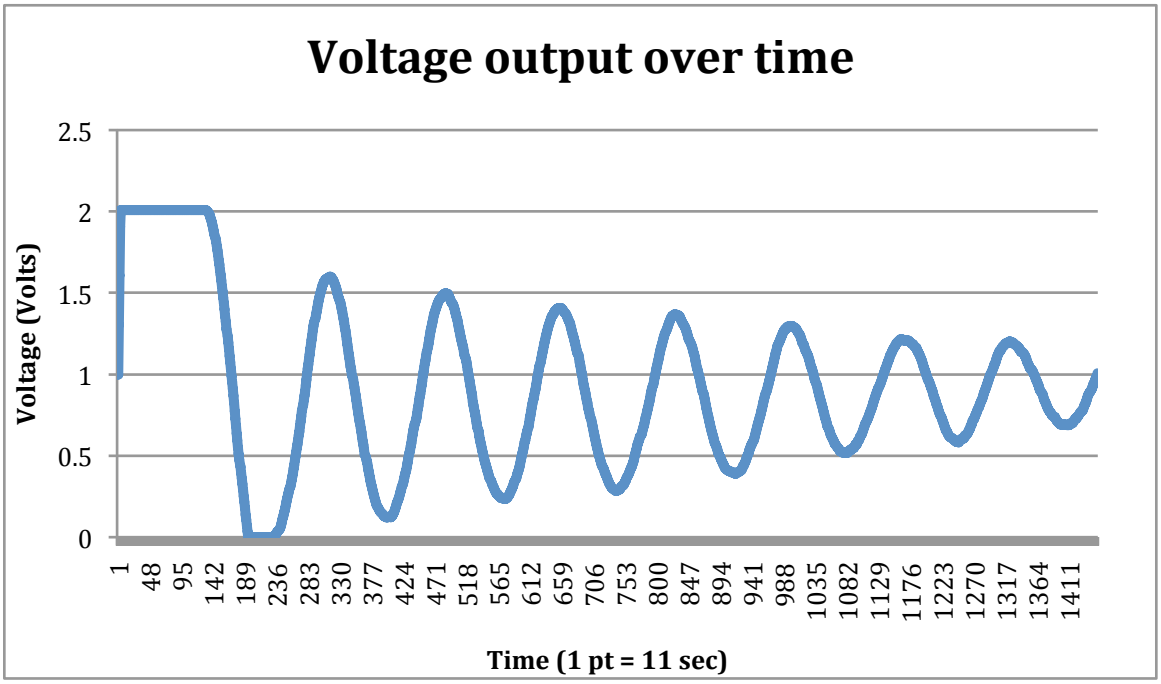
The HEATR's code begins by initializing the EDAS. Then it scans the embedded computer for the analog input module, verifies that it is connected, and initializes it, making sure to allocate memory for the input data received. The program immediately does the same for the analog output module, allocating space on which to write the data to its outputs. After these initial steps, the main segment of the program begins by defining key parameters, specifically the initial output voltage, the desired temperature of the plate, and essential parameters of the input and output data structures. The program reads the initial temperature of the plate from the thermometry circuit, sets values for the proportionality constant of the algorithm and the initial change in output voltage, and enters the servo loop. Within the loop, the output voltage is written to the analog output module and converted from bits to volts to simplify the debugging output. After a ten second pause, the temperature is read from the thermometers 300 times and averaged to reduce noise. The difference between the goal temperature for the plate and this average input temperature is taken and multiplied by the proportionality constant defined before the loop to calculate the necessary change in output voltage. The program outputs the current values of average input voltage and output voltage as a debug string, and the loop

begins again with the new value of change in output voltage, defined to provide negative feedback to the output.

4. Performance and Implementation

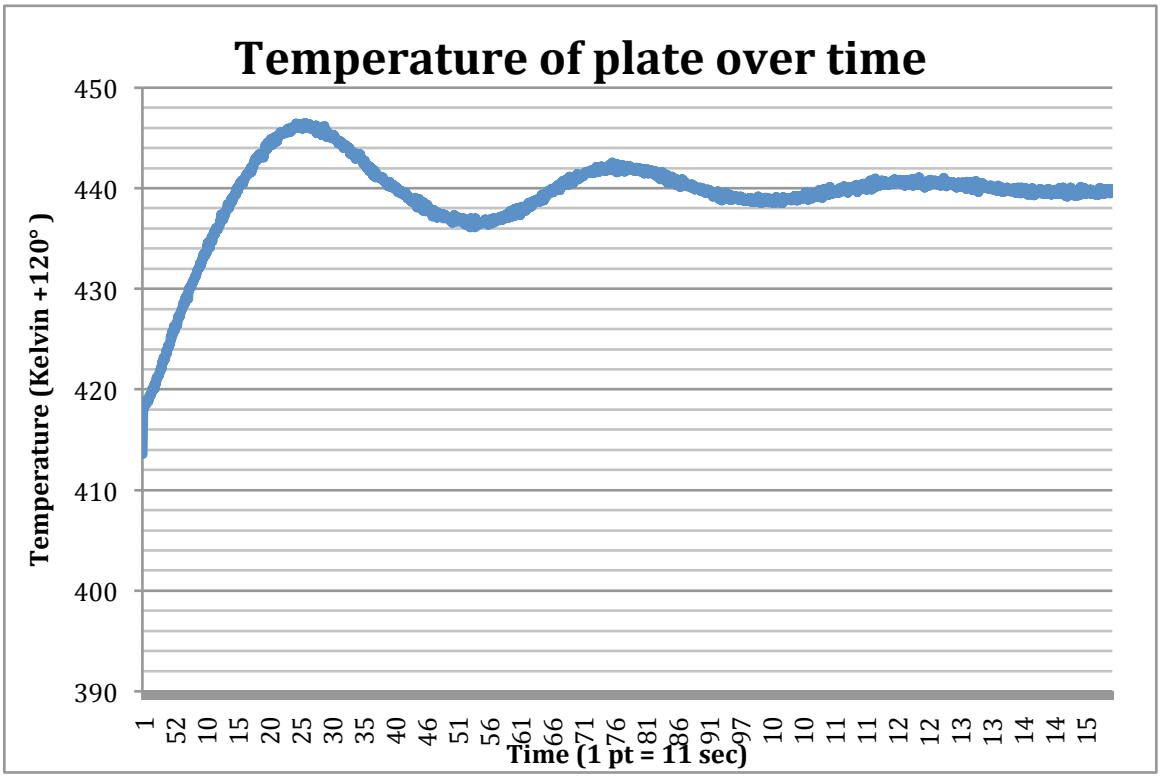
Depending on the value chosen for the proportionality constant, the HEATR can take a long time to raise the plate's temperature to the goal temperature, or it can reach its goal quickly but overshoot it and oscillate about it before reaching a steady value for the output voltage. Testing the program using varying values for the proportionality constant and monitoring the input and output over time enabled me to find the optimal range of values that force the temperature to settle to within one degree of the goal temperature in the least amount of time. Figures 7-12 below show plots of the plate temperature and the voltage output from the EDAS over time for each of the three values of the proportionality constant used. Since the temperature at the summit of Mauna Kea can vary throughout the night, the HEATR system must be able to accommodate a range of differences between the ambient temperature and the goal temperature. For these initial tests with the prototype, we chose a single value for the goal temperature that allowed us to attain a temperature difference close to the maximum expected range during operation at the CSO. The goal temperature set to be approximately 20°C above the ambient temperature in the laboratory. With the constant set to 500.0, the plate achieved a stable temperature within a few degrees in the shortest amount of time, approximately 35 minutes. I did not explore values of Cproportional higher than 500, since we had approximately met our original aim of half-hour stabilization time.

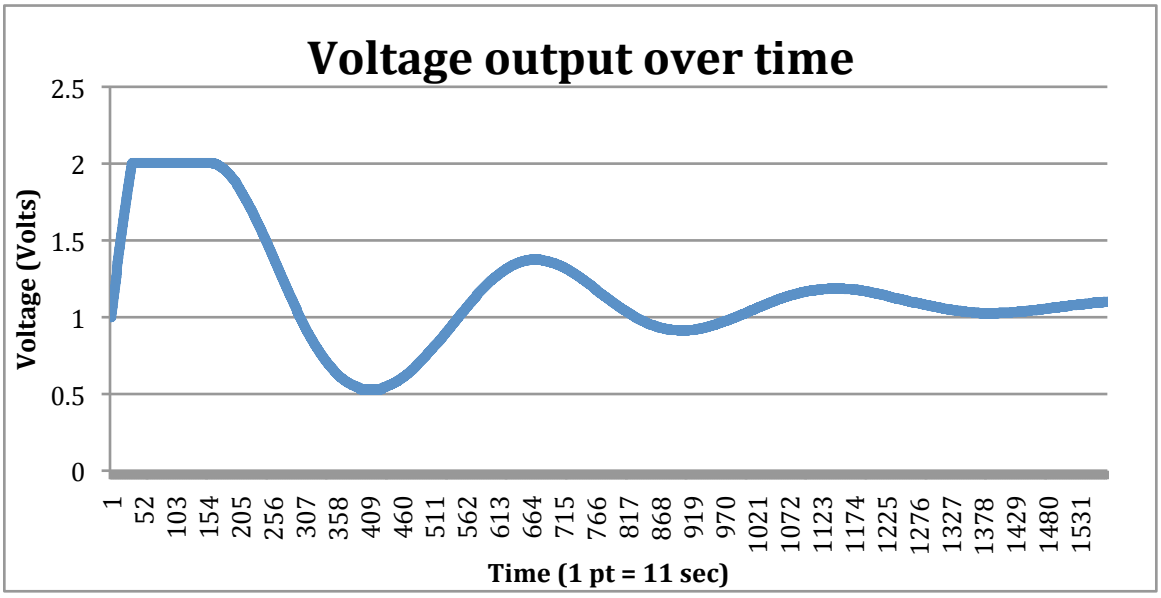




Figures 7 and 8: Performance results for Cproportional = 100.0, tempGoal = 440 (120 K offset + 320 K)

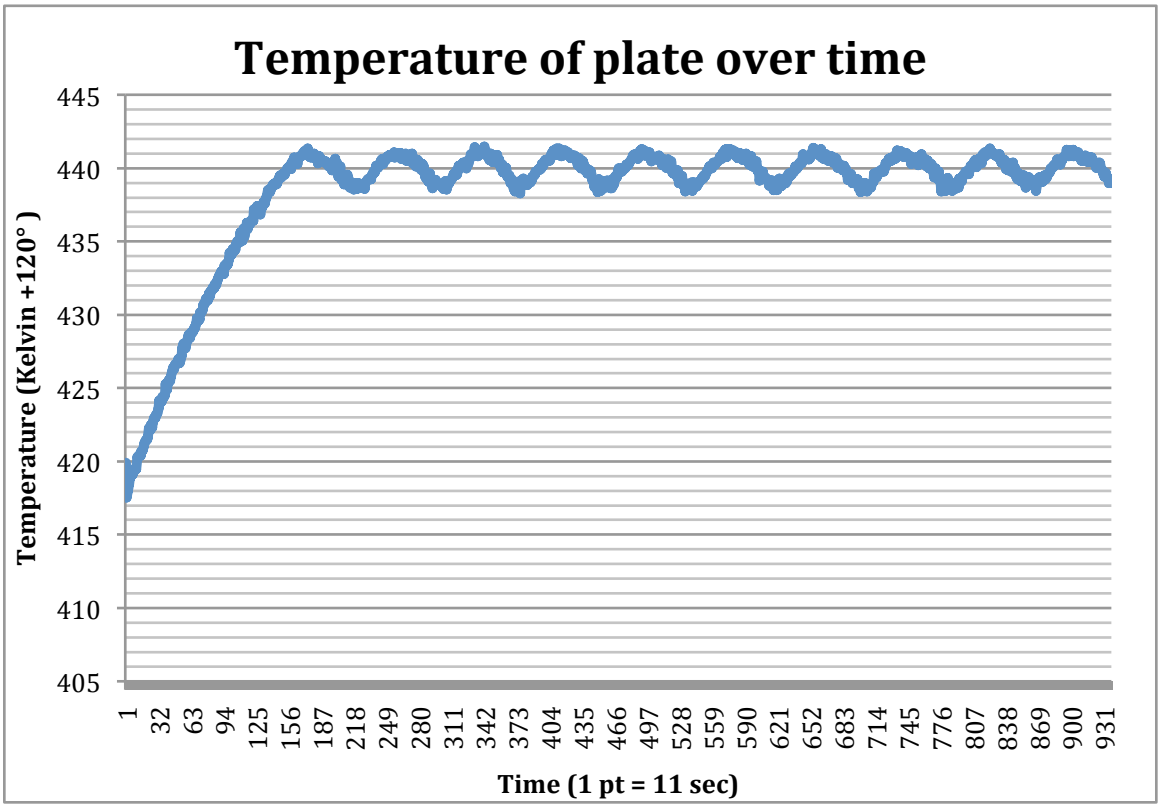
(320 K ≈ 45°C)

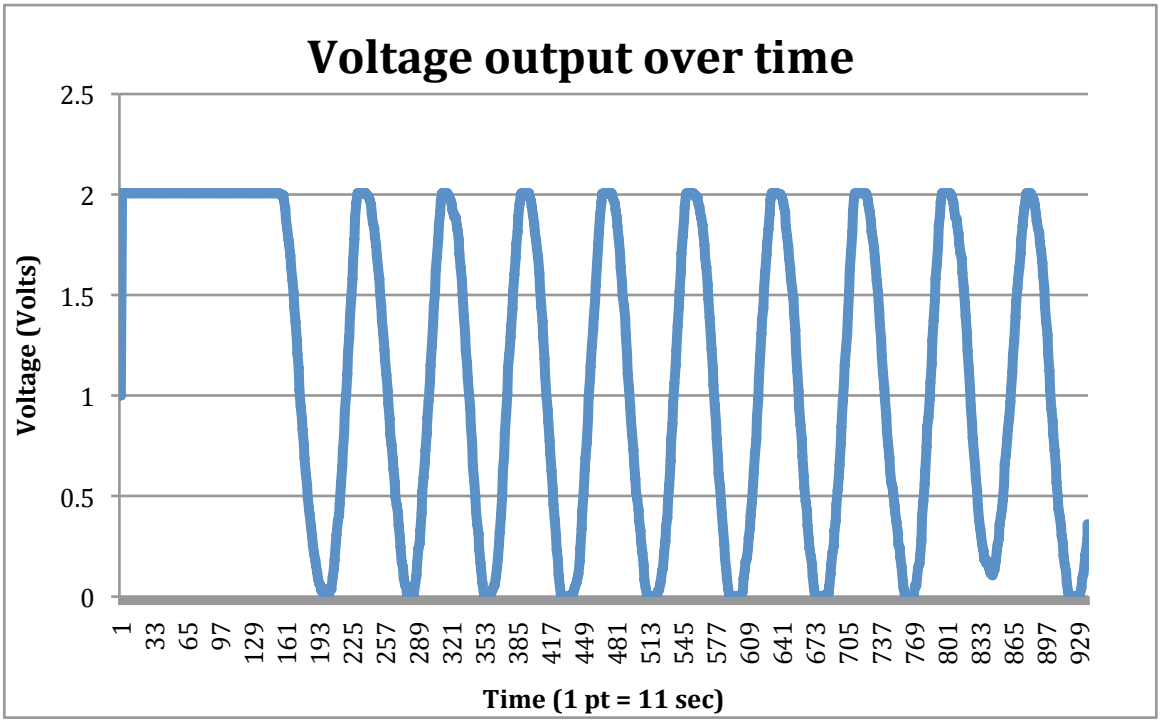




Figures 9 and 10: Performance results for Cproportional = 10.0, tempGoal = 440 (120 K offset + 320 K)

(320 K ≈ 45°C)





Figures 11 and 12: Performance results for Cproportional = 500.0, tempGoal = 440 (120 K offset + 320 K)

(320 K ≈ 45°C)

The time taken to stabilize near the goal temperature, and the stability achieved, can be improved with the implementation of a proportional-integral-derivative controller, or PID controller. In addition to the proportionality term used in the program, the PID controller incorporates a derivative term, which monitors the change between the most recent measurement and the one before it and can act as a damping term if the change is too rapid. Further, the integral term keeps track of a set number of the most recent measurements and ensures that the sum of the errors of these measurements is not too great. In other words, the integral term will increase the output if the last several measurements have all been too low, and it will decrease it if they have been too high. This ensures that any small but consistent error will be corrected. After calibrating the program for the ideal constants of proportionality, integration, and derivation, the PID controller can serve as an even more efficient feedback loop to further limit the amount of time required to reach the goal temperature.

Once a PID loop has been implemented into the code, the HEATR will be ready to be taken to the CSO on Mauna Kea and integrated into SHARP. The power amplifier circuit may be brought out, or a new one may be built according to the schematics included. The resistors mounted to the Box 3 module plate are identical to the ones used to test the system, so this circuit is ready to work with SHARP. Since the plate in the Box 3 module may not be exactly the same mass as the plate used in this test system, the constants of the PID controller may have to be adjusted accordingly. Once optimized, the program will enable observers to run the half-wave plate without worry and instead concern themselves with the task of observing polarized light and mapping magnetic fields.

Appendix A: HEATR Control Program

```
// Mitch_EDAS_AIAOtest.cpp : read and write voltages
//

#include "stdAfx.h"
#include <winbase.h>

// EdasCE definitions. for analog
#include "edasapi.h"
#include "edascons.h"
#include "sysutil.h"

// General EdasCE declarations.
nsOpenSessionReturn OpenSessionReturn;
nsSYSInfoReturn SYSInfoReturn;
nsVERInfoReturn VERInfoReturn;

// Analog Output EDAS-2006 declarations.
int AOcount = 8;
nsAOConfigureData *AOConfigureData;
nsAOWriteData AOWriteData;

// Analog Input EDAS-2003 declarations.
int AIcount = 16;              //originally unsigned
nsAIConfigureData *AIConfigureData;
nsAIReadData AIReadData;
nsAIReadReturn AIReadReturn;

#define kAIGain 1
#define kAIZeroChannel  AI_NO_ZERO
static short datacounts[16];

// Misc. declarations.
#define CR  0x0D
#define LF  0x0A
TCHAR szMessage[255];
int aomoduleid_6;

float sum[50], Vin[50];

int InitRoutine()
{
   long error_code;
   if ((error_code = nsHWInit()) != 0)
   {
      _stprintf(szMessage, _T("Error in nsHWInit()...error=%d\n"), error_code);
      OutputDebugString(szMessage);
      return 1;
   }
   else
   {
      OutputDebugString(_T("nsHWInit() good\n"));
      if ((error_code = nsOpenSession(0, &OpenSessionReturn)) != 0)
      {
         _stprintf(szMessage, _T("Error in nsOpenSession()...error=%d\n"), error_code);
```

```
            OutputDebugString(szMessage);
            return 1;
        }
        else
        {
            OutputDebugString(_T("nsOpenSession() good\n"));
            return 0;
        }
    }
}


void DeInitRoutine()
{
    long error_code;

    if ((error_code = nsCloseSession(OpenSessionReturn.owner, 0, 0)) != 0)
    {
        _stprintf(szMessage, _T("Error in nsCloseSession()...error=%d\n"), error_code);
        OutputDebugString(szMessage);
    }
    else
        OutputDebugString(_T("nscloseSession() good.\n"));
    if ((error_code = nsSWDeinit()) != 0)
    {
        _stprintf(szMessage, _T("Error in nsSWDeinit()...error=%d\n"), error_code);
        OutputDebugString(szMessage);
    }
    else
        OutputDebugString(_T("nsSWDeinit() good.\n"));
}

void ModuleInfo()
{
    long error_code;
    if ((error_code = nsSYSInquire(0, &SYSInfoReturn)) != 0)
    {
        _stprintf(szMessage, _T("Error in nsSYSInquire()...error=%d\n"), error_code);
        OutputDebugString(szMessage);
    }
    else
    {
        _stprintf(szMessage, _T("EdasCE ID=%x\n"), SYSInfoReturn.id);
        OutputDebugString(szMessage);
    }
    if ((error_code = nsVERInquire(0, &VERInfoReturn)) != 0)
    {
        _stprintf(szMessage, _T("Error in nsVERInquire()...error=%d\n"), error_code);
        OutputDebugString(szMessage);
    }
    else
    {
        _stprintf(szMessage, _T("EXECver=%d%c%c"),
                VERInfoReturn.EXECver, CR, LF);
        OutputDebugString(szMessage);
        _stprintf(szMessage, _T("PICver=%x...%c%c"),
```

```
            VERInfoReturn.PICver, CR, LF);
        OutputDebugString(szMessage);
        _stprintf(szMessage,
                _T(" Mod0=%5d... Mod1=%5d... Mod2=%5d... Mod3=%5d%c%c"),
                VERInfoReturn.moduleID[0], VERInfoReturn.moduleID[1],
                VERInfoReturn.moduleID[2], VERInfoReturn.moduleID[3], CR,
                LF);
        OutputDebugString(szMessage);
        _stprintf(szMessage,
                _T(" Mod4=%5d... Mod5=%5d... Mod6=%5d... Mod7=%5d%c%c"),
                VERInfoReturn.moduleID[4], VERInfoReturn.moduleID[5],
                VERInfoReturn.moduleID[6], VERInfoReturn.moduleID[7], CR,
                LF);
        OutputDebugString(szMessage);
        _stprintf(szMessage,
                _T(" Mod8=%5d... Mod9=%5d...Mod10=%5d...Mod11=%5d%c%c"),
                VERInfoReturn.moduleID[8], VERInfoReturn.moduleID[9],
                VERInfoReturn.moduleID[10], VERInfoReturn.moduleID[11],
                CR, LF);
        OutputDebugString(szMessage);
        _stprintf(szMessage,
                _T("Mod12=%5d...Mod13=%5d...Mod14=%5d...Mod15=%5d%c%c"),
                VERInfoReturn.moduleID[12], VERInfoReturn.moduleID[13],
                VERInfoReturn.moduleID[14], VERInfoReturn.moduleID[15],
                CR, LF);
        OutputDebugString(szMessage);
    }
}
```

```
/*****************************************************************
************************  MAIN PROGRAM  ***********************
*****************************************************************/


int WINAPI WinMain(HINSTANCE hInstance,
                HINSTANCE hPrevInstance,
                LPTSTR lpCmdLine,
                int nCmdShow)
{
    // copied from edas2003 (AI) sample program

    long error_code;
    int error;
    unsigned short i;
    int aimoduleid;
    unsigned short x;
            short Vx;
            float Vxv;
            float Vin;
            float tempGoal;
```

```
            float HeaterGuess;
            float deltaVproportional;
            float Cproportional;
            int nread;
            float Vsum;

   OutputDebugString(_T("Mitchell's EdasCE AI-AO Test Program initialized.\n"));
   error = InitRoutine();
   if (error == 1)
   {
      _stprintf(szMessage, _T("EdasCE Initialization error.\n"));
      OutputDebugString(szMessage);
   }

   else
   {
      ModuleInfo();

/***********************************Initialize Analog Input Module ******************/

/* Find an EDAS analog input module. */
      aimoduleid = -1;
      for (i = 0; i < 16; i++)
      {
         if (VERInfoReturn.moduleID[i] == 131)
         {
            aimoduleid = i;
            _stprintf(szMessage, _T("Module %d is a Analog ESAS-2003M-2/3.\n"), aimoduleid);
            OutputDebugString(szMessage);
         }
         if (VERInfoReturn.moduleID[i] == 3)
         {
            aimoduleid = i;
            _stprintf(szMessage, _T("Module %d is a Analog ESAS-2003M-2/3.\n"), aimoduleid);
            OutputDebugString(szMessage);
         }
      }
      if (aimoduleid >= 0)
      {
//  Allocate memory for the Analog input channel list.
//         error = i3malloc(sizeof(nsAIConfigureData) + AIcount * sizeof(nsAIList), (void
**)&AIConfigureData);
         error = i3malloc(sizeof(short) + AIcount * sizeof(nsAIList), (void **) &AIConfigureData);
// Configure the Channel List.
         AIConfigureData->count = AIcount;
         for (i = 0; i < AIcount; i++)
         {
            AIConfigureData->list[i].module = aimoduleid;
            AIConfigureData->list[i].channel = (unsigned short) i;
            AIConfigureData->list[i].gain = 1;
            AIConfigureData->list[i].range = BIPOLAR_20;
            AIConfigureData->list[i].diff = 0;
            AIConfigureData->list[i].zero = kAIZeroChannel;
         }
         if ((error_code = nsAIConfigureList(OpenSessionReturn.owner,
                        AIConfigureData, 0)) != 0) {
```

```
            _stprintf(szMessage,
                    _T("nsAIConfigureData()...error_code=%d\n"),
                    error_code);
            OutputDebugString(szMessage);
         }
         else
         {
            _stprintf(szMessage, _T("nsAIConfigureList OK.\n"));
            OutputDebugString(szMessage);

         }
      }

/**********************Initialize Analog Output Module ****************************/

/* Find an EDAS analog output module. */

      for (i = 0; i < 16; i++)
      {
         if (VERInfoReturn.moduleID[i] == 6)
         {
            aomoduleid_6 = i;
            _stprintf(szMessage, _T("Module %d is an Analog Output EDAS-2006M-1.%c%c"),
aomoduleid_6, CR, LF);
            OutputDebugString(szMessage);
         }
      }
// If EDAS-2006 is present, configure the Analog Output Module.
      if (aomoduleid_6 >= 0)
      {
// Allocate Memory for the AOConfigureData() structure.
         if ((error_code = i3malloc(sizeof(nsAOConfigureData) + (AOcount * sizeof(nsAOList)),
                  (void **) &AOConfigureData)) != 0)
         {
            _stprintf(szMessage, _T("ERROR from i3malloc()...error_code=%d\n"), error_code);
            OutputDebugString(szMessage);
         }
         else
         {
            AOConfigureData->count = AOcount;
            for (i = 0; i < AOcount; i++)
            {
               AOConfigureData->list[i].module = aomoduleid_6;
               AOConfigureData->list[i].channel = (unsigned short) i;
               AOConfigureData->list[i].range = UNIPOLAR_10;
            }
            if ((error_code =
                nsAOConfigureList(OpenSessionReturn.owner, AOConfigureData, 0)) != 0)
                {
               _stprintf(szMessage, _T ("EDAS-2006...nsAOConfigureData()...error_code=%d%c%c"),
error_code, CR, LF);
               OutputDebugString(szMessage);
            }
            else
            {
               _stprintf(szMessage, _T("EDAS-2006...nsAOConfigureList OK.%c%c"), CR, LF);
```

```
            OutputDebugString(szMessage);
        }

    }

                /************************** TEST PROGRAM AI-AO
**********************************/

                        AOWriteData.module = aomoduleid_6;
                        HeaterGuess = 6553.5;           //Sets initial output voltage to 1 volt (in bits);
65535 = 10V
                        tempGoal = 440.0;               //Sets desired temperature in Kelvin
                        Vx = 6553.5;
                        x = 0;                          //for AIReadReturn.data structure; set to 0 since
AIReadData.count = 1


                        AIReadData.module = aimoduleid;
                        AIReadData.channel = 4;         //Sets channel being read (starting with
channel 0)
                        AIReadData.count = 1;           //Sets number of times channel is read at once
                        AIReadData.tflag = 0;

                        AOWriteData.data[0] = HeaterGuess;  //Writes Vx to output module
                        AOWriteData.channel = 1;        //Sets channel being written to
                        Vxv = HeaterGuess/6553.5;       //Converts output in bits to output in volts
                        _stprintf(szMessage, _T("Outputting Vx = %5.3f volts. Ready!\n"), Vxv);
                        OutputDebugString(szMessage);
                        Sleep(10000);

                        if ((error_code = nsAIRead(OpenSessionReturn.owner, &AIReadData,
&AIReadReturn)) != 0)
                        {
                                _stprintf(szMessage, _T("nsAIRead()...error_code=%d\n"),
error_code);
                                OutputDebugString(szMessage);
                        }
                        else if ((error_code = nsAOWrite(OpenSessionReturn.owner, &AOWriteData,
NULL)) != 0)
                        {
                                _stprintf(szMessage, _T("nsAOWriteData()...error_code=%d\n"),
error_code);
                                OutputDebugString(szMessage);
                        }
                        else
                        {
                                Vin = (((float)(AIReadReturn.data[x]>>4))/2048.0)*1000.0; //last
number was originally *10.0; changed to match scale
                        }
                        _stprintf( szMessage, _T("Vin    Vout\n"));
                        OutputDebugString(szMessage);
                        _stprintf( szMessage, _T("%5.3f %5.3f\n"), Vin, Vxv);
                        OutputDebugString(szMessage);

                        deltaVproportional = 0.0;
                        Cproportional = 100.0;
```

```
                        while (Vin < 10000.0)                   //loop ends if input voltage exceeds 9V
                        {
                                Vx = Vx + deltaVproportional;    //new output voltage using input
voltage
                                if (Vx > 13150.0)
                                {
                                        Vx = 13150.0;
                                }
                                else if (Vx < 0.0)
                                {
                                        Vx = 0.0;
                                }
                                else
                                {
                                        Vx = Vx;
                                }
                                Vxv = Vx/6553.5;                //convert output from bits to volts

                                Sleep(1000);
                                AOWriteData.data[0] = Vx;           //write new output voltage to
output module
                                if ((error_code = nsAOWrite(OpenSessionReturn.owner,
&AOWriteData, NULL)) != 0)
                                {
                                        _stprintf(szMessage,
_T("nsAOWriteData()...error_code=%d\n"), error_code);
                                        OutputDebugString(szMessage);
                                }
                                Sleep(10000);
                                Vsum = 0.0;
                                for (nread = 1; nread <= 300; nread++)  //take 300 voltage readings and
average
                                {
                                        if ((error_code = nsAIRead(OpenSessionReturn.owner,
&AIReadData, &AIReadReturn)) != 0)
                                        {
                                                _stprintf(szMessage,
_T("nsAIRead()...error_code=%d\n"), error_code);
                                                OutputDebugString(szMessage);
                                        }
                                        else
                                        {
                                                Vin =
(((float)(AIReadReturn.data[x]>>4))/2048.0)*1000.0; //get input voltage; last number was originally *10.0
                                        }
                                        Vsum = Vsum + Vin;
                                }
                                Vin = Vsum / 300;    //average Vsum
                                deltaVproportional = -1.0*(Vin - tempGoal) * Cproportional;

                                _stprintf(szMessage, _T("%5.3f %5.3f\n"), Vin, Vxv);
                                OutputDebugString(szMessage);
                        }
        }
        else
```

```
        {
            OutputDebugString(_T("Unable to detect a EDAS-2006 Analog Module...terminate program.\n"));
        }

    }

    // Free memory allocation for channel list configuration.

    i3free(AIReadReturn.data);

    i3free(AOConfigureData);
    i3free(AIConfigureData);


    DeInitRoutine();

    return 0;
}
```

REFERENCES

1. R. L. Hotz. "Scientists Uncover Evidence of New Planets Orbiting Star." *The Tech Online Edition*. The Tech, 22 April 1994. Web. 5 May 2012.

2. *PlanetQuest: The Search for Another Earth*. NASA Jet Propulsion Laboratory. Web. 5 May 2012.

3. *Kepler: A Search for Habitable Planets*. NASA Ames Research Center. Web. 6 May 2012.

4. S. W. Stahler. "The Early Life of Stars." *Scientific American* July 1991: 48-51. Print.

5. E. N. Parker. "Magnetic Fields in the Cosmos." *Scientific American* August 1983: 44-49. Print.

6. B. T. Draine. *Physics of the Interstellar and Intergalactic Medium*. Princeton, NJ: Princeton University Press, 2011. Print.

7. M. Mac Low, R. S. Klessen. "Control of Star Formation by Supersonic Turbulence." Review of Modern Physics Vol. 76, 125-194. January 2004.

8. R. H. Hildebrand et al. "A Primer on Far-Infrared Polarimetry." Publications of the Astronomical Society of the Pacific Vol. 112, No. 775, 1215-1235. September 2000.

9. H. Li et al. "Design and initial performance of SHARP, a polarimeter for the SHARC-II camera at the Caltech Submillimeter Observatory 2008." Applied Optics Vol. 47, No. 3, 422-430. 2008.

10. "Needle Thrust Bearings and Washers from National Precision." *National Precision Bearing*. National Precision Bearing, Inc., 2012. Web. 7 May 2012.