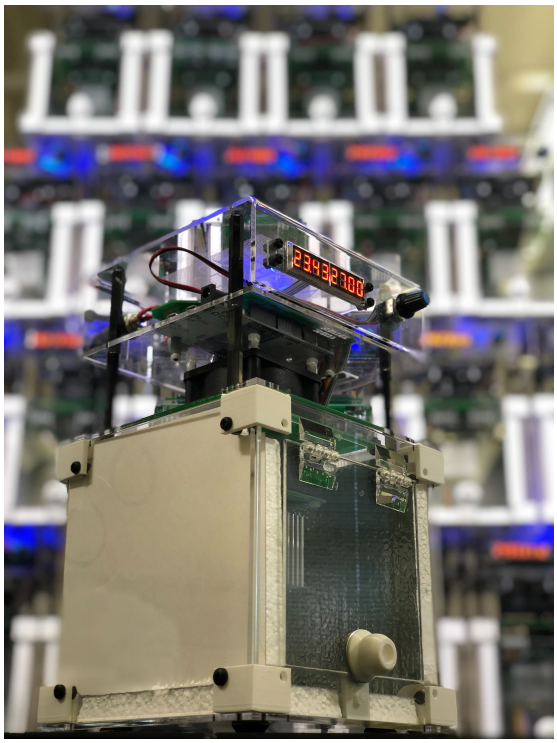
Ajoy PID Temperature Control
Operation & Implementation

Sam Johnson

December 2020



University of California, Berkeley
Berkeley Ajoy NMR Lab, Stanley Hall

Contents

1 Product Description

1.1 Market Need

Climate Controlled storage is an emerging issue that has seen a recent growth in market capitalization. Typically only available to large companies with extensive purchasing power, temperature controlled environments provide safe and secure climate for items including, but not limited too: hardware, biological samples, perishables, rare artwork, books, electronics, fungi-culture, and growing bacteria. As a hobbyist individual, one could benefit from such a device, however none are currently on the market within an affordable price range. This product will provide a clean, portable, and effective temperature controlled climate for a small 7" x 7" x 7" space with the low cost of under $100. Using simple hardware and components found online, as well as a introductory programming experience, this product can be available to hobbyists and budget scientists alike.

1.2 Academic Need

In an academic setting, a hands on lab kit such as this serves a great method of engagement and concept mastery. For an engineering student, the project combines hardware design, circuitry, mechatronics, and programming, as well as data acquisition and an introduction to control theory, specifically PID control. All of these skills are extremely valuable as an undergraduate, and the ability to correctly interface hardware and software together is a valuable tool set that will provide any university student with the knowledge necessary to design, manufacture, and control a project from the ground up. Ideally, this project could be boxed and distributed as a university laboratory kit for undergrads, and with its relatively low cost and hands on assembly, it would encompass a plethora of material.

1.3 The PID Box Solution

The PID Box accomplishes all of these goals, while introducing a fun and exciting project. Utilizing primarily a mixture of LabVIEW and Arduino, this device will be able to regulate temperature in the range of 10 - 35 Degrees Celsius. It will allow the user to define a target temperature, and then using a tuned PID controller, the box will drive the temperature to the set point through the use of internal heating, internal and external cooling, and a variety of control commands that can be sent to the actuator. There is also a clean user interface that will allow the owner to visualize the state of the box in real time, as well as further implement changes as they see fit. Using a different set of sensors and actuators, this box can be further tuned and expanded upon to increase efficiency, accuracy, and operating size. This low-cost home project is good for industry professionals, researchers, and students alike.

2 Design

2.1 Bill of Materials

Part #	Component	Quantity	Cost ($)
1	Aluminum Heat Sink Cooler Fin	2	3.99
2	Arduino Due	1	18.00
3	12V DC Cooling Fan	2	4.75
4	TMP275 Temperature Sensor	2	3.50
5	T115 Thermistor	1	1.35
6	Custom Temperature Sensor PCB	2	3.00
7	Custom Heat Exchanger Passthrough PCB	1	2.25
8	Custom Heater Mount PCB	1	2.25
9	Custom Heater Fin Connector PCB	1	2.25
10	Custom Arduino Shield PCB	1	3.50
11	6”x 6”x 3/16 High Strength Acrylic	1	3.37
12	Polyurethane Insulation Sheet	1	8.00
13	30 cm Breadboard Jumper Wires	1	5.70
14	16 ft 30 Pin Ribbon Cable	1	8.99
15	#1-72 x 1/8 Nylon Screws	34	3.65
16	#1-72 x 1/8 Nylon Screws	34	3.65
17	10 Pin Ribbon Cable Connector	4	1.30
18	12 V Rocker Switch	1	0.95
19	20 mA Light Emitting Diode	3	0.40
20	3.6x10 Glass Slow Blow Fuse	1	0.05
21	8-Digital LED Segment Display	1	4.80
22	5V A23 Rotary Encoder Potentiometer	1	1.88
23	Micro USB Cable	1	0.18
24	12V DC Power Supply	1	0.50
25	Rosin Core Solder	1	3.25
26	1.75 mm PLA Filament	1	0.85
		Total	$ 92.36

2.2 Assembly, Schematic & Render

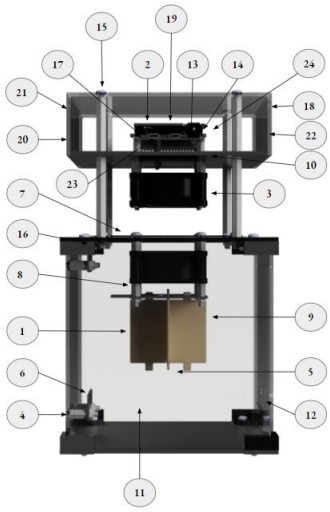


Figure 1: Labeled Render [See BOM]

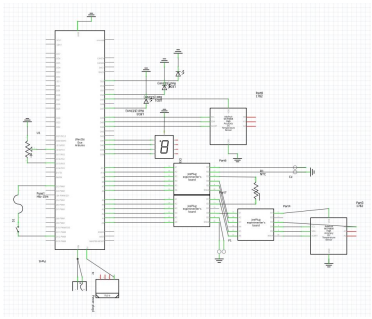


Figure 2: Labeled Render [See BOM]

3 Operation

3.1 Arduino IDE

The software implementation is split into two categories: front end and back end. The foundation of the PID box is programmed in Arduino [Appendix A], which utilizes a simple code structure and easy Arduino board integration to actuate the box with ease. Connected to an Arduino Due, the board is further connected to multiple custom made PCB's which rout power and control to separate components on the board. The primary driver of the system is the set of command phrases. After querying in terminal, the user can write commands such as "HEAT ON", "HEAT OFF", "FANCOOL ON" and multiple others that, when entered, have a corresponding section of code executed. For example, the "HEAT ON" command, written as a string, directs the corresponding pin to regulate the voltage needed to activate the heater, similar to the two DC fans on board. Due to the complexity of the assembly, a variety of pins are used to control the device, subsequently using an array of ribbon cables and PCB's to organize the assortment of electronic components. Portions of the device were assembled prior, such as the soldering of temperature sensors, PCB connectors, and heat exchangers. The primary structure of the code is a set of if statements, effectively polling for the execution of a targeted command. There are Three primary commands programmed into the Arduino control scheme. *Heat Fast* enables both the internal heater and the internal fan, while also disabling the external fan. *Idle Fast* disables the heater, enables the internal fan, and disables the outer fan. Finally, *Cool Fast* disables the heater, and enables both the internal and external fan. These commands, combined with the correct operation time (Time Constant) allow the state of the box to be accurately driven.

3.2 LabVIEW

LabVIEW implementation provides the front end of the program. Using a custom made VI, the user is presented with a clean virtual interface that can be used to track, visualize, and control the target temperature of the box all on one page. The primary tool being used in this design is Serial communication, where LabVIEW interacts with the box through a COM port, using only Serial Read and Serial Write. On startup, the user is presented with a blank interface [Appendix B]. From the drop down menu, the user has the option to select available COM ports, choose a target temperature, and also set the Y-Axis bounds on the graph for easier analysis. When the program is initialized, the box is still in an idling state because no commands have been sent. LabVIEW first utilizes serial write to send a "TEMPIN?" command to the Arduino, from which the Arduino code on the Due will return the temperature in the "xx.xx" format. The same is done for "TEMPOUT?". After this, LabVIEW plots the temperatures on a real time waveform chart, updating every 150 ms using this method. As well as plotting inside and outside temperatures, the program also

calculates error deviation from the setpoint. Following this, the PID controller can now take over. Calculating the change in values (Derivative), summing the cumulative error (Integral), and adding the deviation from the setpoint (Proportional), software is used to calculate the correct command to send back to the arduino, as well as the period of time for which it should be active before further querying for temperature and continuing the process.

3.3 Hardware

There is an abundance of hardware being used in this device, however a brief overview of component and related use will be given. The **Arduino Due** is the micro-controller that directs the operation of the box. It communicates with the computer and LabVIEW software, as well as all of the electronics on board. The **PCB** are used to house specific components, such as the sensors, DC fans, potentiometer, and LED Display while providing support, neatness, accuracy, and expanded circuitry using ribbon cables. The **Temperature Sensors** are used to read and transmit temperature back to the Arduino once they are queried. These are located both on the inside and outside of the box. There are two **DC Fans** that are places similarly on the inside and outside of the container, and are used to regulate temperature. The heating is accomplished through the use of a **Thermistor** and two parallel **Aluminum Fins** which provide a controlled rate of convective and radiative heat transfer when current is applied to each of the inner surfaces. The **Insulating Foam** is used to contain heat and prevent variation in temperature, and the **LED Display** is used to display the queried temperatures in the form "Outside, Inside". Finally, **Ribbon Cables** are used as internal connections, and connect all components in the device, as well as the heat exchanger passthrough and adjacent PCB's.

3.4 Finite State Diagram

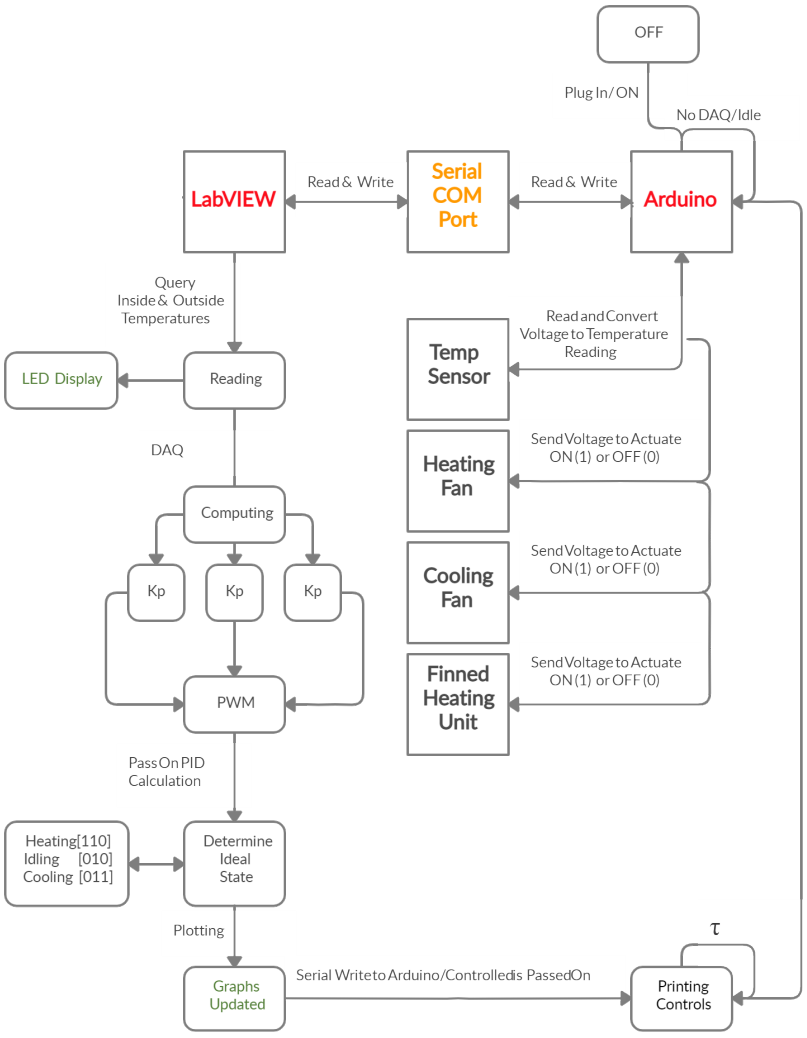


Figure 3: Finite State Diagram

4 Appendix A: Arduino IDE

```
#include "LedControl.h"
#include "BitBang_I2C_due_compat.h"
// Note: This library is modified to be compatible with the Arduino Due, the
//latest version at the time of writing this: 2.1.3 has a bug when used with
//the Due. Lines 558 through 577 need to be commented out.

#define SDA_PIN 2
#define SCL_PIN 3
#define SDA_PIN_IN 47
#define SCL_PIN_IN 45
#define COMMAND_NUM 16

BBI2C bbi2c;
BBI2C bbi2c_in;
LedControl lc = LedControl(59,60,58,1);
int verbose_mode = 0;

void setup() {
  // GPIO init
  pinMode(13, OUTPUT); //FAN_COOL_EN
  pinMode(25, OUTPUT); //HEAT EN
  pinMode(23, OUTPUT); //FAN_HEAT_EN
  pinMode(30, OUTPUT); //GP LED

  // LED display init
  lc.shutdown(0,false);
  lc.setIntensity(0,8);
  lc.clearDisplay(0);

  pinMode(61, INPUT); //Rotary encoder D0
  pinMode(65, INPUT); //Rotary encoder D1
  pinMode(66, INPUT); //Rotary encoder switch

  // Outside temperature
  Serial.begin(115200);
  SerialUSB.begin(115200);
//  SerialUSB.begin(1843200);
  memset(&bbi2c, 0, sizeof(bbi2c));
  bbi2c.bWire = 0; // use bit bang, not wire library
  bbi2c.iSDA = SDA_PIN;
  bbi2c.iSCL = SCL_PIN;
  I2CInit(&bbi2c, 1000000);

  // Inside temperature
  memset(&bbi2c_in, 0, sizeof(bbi2c_in));
  bbi2c_in.bWire = 0; // use bit bang, not wire library
  bbi2c_in.iSDA = SDA_PIN_IN;
  bbi2c_in.iSCL = SCL_PIN_IN;
  I2CInit(&bbi2c_in, 1000000);
  delay(100); // allow devices to power up

  // set to 12bit temperature
  uint8_t configreg[]={0x01,0x60};
```

```
  I2CWrite(&bbi2c, 0x48, configreg, 2);
  I2CWrite(&bbi2c_in, 0x48, configreg, 2);
}

void displaytempertures(){
  float temp_val = 0;
  uint8_t temp[2];
  I2CReadRegister(&bbi2c, 0x48, 0x00, temp, 2);
  String tempstringH = String(temp[0]);
  String tempstringL = String(temp[1]);
  //          Serial.println(tempstringH);
  //          Serial.println(tempstringL);
  temp_val = ((temp[1]>>4)&0x0F)*0.0625;
  temp_val = temp_val + ((temp[0]&0x7F)*1.0);
  if (temp[0]&0x80){
    temp_val = -1.0*temp_val;
  }
  String tempcel = String(temp_val);
  Serial.println("temperature in celcius:");
  Serial.println(tempcel);
  lc.setDigit(0,7,(int(temp_val/10)%10),false);
  lc.setDigit(0,6,(int(temp_val/1)%10),true);
  lc.setDigit(0,5,(int(temp_val*10)%10),false);
  lc.setDigit(0,4,(int(temp_val*100)%10),false);

  float temp_val_in = 0;
  uint8_t temp_in[2];
  I2CReadRegister(&bbi2c_in, 0x48, 0x00, temp_in, 2);
  String tempstringH_in = String(temp_in[0]);
  String tempstringL_in = String(temp_in[1]);
//          Serial.println(tempstringH_in);
//          Serial.println(tempstringL_in);
  temp_val_in = ((temp_in[1]>>4)&0x0F)*0.0625;
  temp_val_in = temp_val_in + ((temp_in[0]&0x7F)*1.0);
  if (temp_in[0]&0x80){
    temp_val_in = -1.0*temp_val_in;
  }
  String tempcel_in = String(temp_val_in);
  Serial.println("temperature in celcius inside box:");
  Serial.println(tempcel_in);
  lc.setDigit(0,3,(int(temp_val_in/10)%10),false);
  lc.setDigit(0,2,(int(temp_val_in/1)%10),true);
  lc.setDigit(0,1,(int(temp_val_in*10)%10),false);
  lc.setDigit(0,0,(int(temp_val_in*100)%10),false);

  SerialUSB.println(tempcel + "," + tempcel_in);

  delay(100);
  return;
}

void displaynumber(int display_number){
  lc.setDigit(0,7,(int(display_number/10000000)%10),false);
  lc.setDigit(0,6,(int(display_number/1000000)%10),false);
  lc.setDigit(0,5,(int(display_number/100000)%10),false);
  lc.setDigit(0,4,(int(display_number/10000)%10),false);

```

```
  lc.setDigit(0,3,(int(display_number/1000)%10),false);
  lc.setDigit(0,2,(int(display_number/100)%10),false);
  lc.setDigit(0,1,(int(display_number/10)%10),false);
  lc.setDigit(0,0,(int(display_number/1)%10),false);
  delay(100);
  return;
}


int command_parse(int *read_buffer){
  const char *list_commands[COMMAND_NUM];
  list_commands[0]="?\r";
  list_commands[1]="HELP\n";
  list_commands[2]="HEAT ON\n";
  list_commands[3]="HEAT OFF\n";
  list_commands[4]="FANHEAT ON\n";
  list_commands[5]="FANHEAT OFF\n";
  list_commands[6]="FANCOOL ON\n";
  list_commands[7]="FANCOOL OFF\n";
  list_commands[8]="TEMPIN?\n";
  list_commands[9]="TEMPOUT?\n";
  list_commands[10]="DISPLAY TEMP\n";
  list_commands[11]="GREEN ON\n";
  list_commands[12]="GREEN OFF\n";
  list_commands[13]="DISPLAY 8DIGIT\n";
  list_commands[14]="VERBOSE OFF\n";
  list_commands[15]="VERBOSE ON\n";

  uint match_mask = 0xFFFF;
  int done = 0;
  int command_number_result = -1;

  for(int i=0; done == 0;i++){
    for(int j=0;j<COMMAND_NUM;j++){
      if((read_buffer[i] == 13)||(read_buffer[i] == 10)){
        if((list_commands[j][i] == 13)||(list_commands[j][i] == 10)
    ){
        }else{
          match_mask = match_mask & ~(1<<j);
        }
        done =1;
      }else if(list_commands[j][i]==read_buffer[i]){
      }else{
        match_mask = match_mask & ~(1<<j);
      }
    }
  }
  for(int i=0;i<COMMAND_NUM;i++){
    if(match_mask&(1<<i)){
      command_number_result = i;
      break;
    }
  }
  return command_number_result;
}

void string_inside_temp(char *inside_temp_string){
```

```
  float temp_val_in = 0;
  uint8_t temp_in[2];
  String temp_val_in_string;
  I2CReadRegister(&bbi2c_in, 0x48, 0x00, temp_in, 2);
  if(temp_in[0]==0&temp_in[1]==0){
    I2CReadRegister(&bbi2c_in, 0x49, 0x00, temp_in, 2);
    if(temp_in[0]==0&temp_in[1]==0){
      I2CReadRegister(&bbi2c_in, 0x4A, 0x00, temp_in, 2);
      if(temp_in[0]==0&temp_in[1]==0){
        I2CReadRegister(&bbi2c_in, 0x4B, 0x00, temp_in, 2);
        if(temp_in[0]==0&temp_in[1]==0){
          I2CReadRegister(&bbi2c_in, 0x4C, 0x00, temp_in, 2);
          if(temp_in[0]==0&temp_in[1]==0){
            I2CReadRegister(&bbi2c_in, 0x4D, 0x00, temp_in, 2);
            if(temp_in[0]==0&temp_in[1]==0){
              I2CReadRegister(&bbi2c_in, 0x4E, 0x00, temp_in, 2);
              if(temp_in[0]==0&temp_in[1]==0){
                I2CReadRegister(&bbi2c_in, 0x4F, 0x00, temp_in, 2);
                if(temp_in[0]==0&temp_in[1]==0){
                  I2CReadRegister(&bbi2c_in, 0x48, 0x00, temp_in,
    2);
                }
              }
            }
          }
        }
      }
    }
  }
  String tempstringH_in = String(temp_in[0]);
  String tempstringL_in = String(temp_in[1]);
  temp_val_in = ((temp_in[1]>>4)&0x0F)*0.0625;
  temp_val_in = temp_val_in + ((temp_in[0]&0x7F)*1.0);
  if (temp_in[0]&0x80){
    temp_val_in = -1.0*temp_val_in;
  }
  temp_val_in_string = String(temp_val_in);
  temp_val_in_string.toCharArray(inside_temp_string,15);
  return;
}

void string_outside_temp(char *inside_temp_string){
  float temp_val_in = 0;
  uint8_t temp_in[2];
  String temp_val_in_string;
  I2CReadRegister(&bbi2c, 0x48, 0x00, temp_in, 2);
  if(temp_in[0]==0&temp_in[1]==0){
    I2CReadRegister(&bbi2c, 0x49, 0x00, temp_in, 2);
    if(temp_in[0]==0&temp_in[1]==0){
      I2CReadRegister(&bbi2c, 0x4A, 0x00, temp_in, 2);
      if(temp_in[0]==0&temp_in[1]==0){
        I2CReadRegister(&bbi2c, 0x4B, 0x00, temp_in, 2);
        if(temp_in[0]==0&temp_in[1]==0){
          I2CReadRegister(&bbi2c, 0x4C, 0x00, temp_in, 2);
          if(temp_in[0]==0&temp_in[1]==0){
            I2CReadRegister(&bbi2c, 0x4D, 0x00, temp_in, 2);
            if(temp_in[0]==0&temp_in[1]==0){
```

```
223               I2CReadRegister(&bbi2c, 0x4E, 0x00, temp_in, 2);
224               if(temp_in[0]==0&temp_in[1]==0){
225                 I2CReadRegister(&bbi2c, 0x4F, 0x00, temp_in, 2);
226                 if(temp_in[0]==0&temp_in[1]==0){
227                   I2CReadRegister(&bbi2c, 0x48, 0x00, temp_in, 2);
228                 }
229               }
230             }
231           }
232         }
233       }
234     }
235   }
236   String tempstringH_in = String(temp_in[0]);
237   String tempstringL_in = String(temp_in[1]);
238   temp_val_in = ((temp_in[1]>>4)&0x0F)*0.0625;
239   temp_val_in = temp_val_in + ((temp_in[0]&0x7F)*1.0);
240   if (temp_in[0]&0x80){
241     temp_val_in = -1.0*temp_val_in;
242   }
243   temp_val_in_string = String(temp_val_in);
244   temp_val_in_string.toCharArray(inside_temp_string,15);
245   return;
246 }
247
248
249 void command_execute(int command_number){
250 //  list_commands[0]="?\r";
251 //  list_commands[1]="HELP\n";
252 //  list_commands[2]="HEAT ON\n";
253 //  list_commands[3]="HEAT OFF\n";
254 //  list_commands[4]="FANHEAT ON\n";
255 //  list_commands[5]="FANHEAT OFF\n";
256 //  list_commands[6]="FANCOOL ON\n";
257 //  list_commands[7]="FANCOOL OFF\n";
258 //  list_commands[8]="TEMPIN?\n";
259 //  list_commands[9]="TEMPOUT?\n";
260 //  list_commands[10]="DISPLAY TEMP\n";
261 //  list_commands[11]="GREEN ON\n";
262 //  list_commands[12]="GREEN OFF\n";
263 //  list_commands[13]="DISPLAY 8DIGIT\n";
264 //  list_commands[14]="VERBOSE OFF\n";
265 //  list_commands[15]="VERBOSE ON\n";
266
267   const char *help_text_var = "PID temperature box commands:\n\
      nCommand: \"HELP\
268   \" or \"?\"\nResponse: Displays all commands\nDescription: Prints
        this \
269   list of commands\n\nCommand: \"HEAT ON\"\nResponse: \"Turning
      heater on\
270   \"\nDescription: Turns heater on\n\nCommand: \"HEAT OFF\"\
      nResponse: \"Turning\
271   heater off\"\nDescription: Turns heater off\n\nCommand: \"TEMPIN
      ?\"\n\
272   Response: \"<temperature>\"\nDescription: Queries temperature
      inside the box,\
```

```
  and returns temperature in celcius and in decimal notation,
    example \"10.2\"\
\n\nCommand: \"TEMPOUT?\"\nResponse: \"<temperature>\"\
    nDescription: Queries\
temperature outside the box, and returns temperature in celcius
    and in decimal\
notation, example \"10.2\"\n\nCommand: \"FANHEAT ON\"\nResponse:
    \"Turning\
heater fan on\"\nDescription: Turns heater fan on\n\nCommand: \"
    FANHEAT OFF\"\
\nResponse: \"Turning heater fan off\"\nDescription: Turns heater
     fan off\n\n\
Command: \"FANCOOL ON\"\nResponse: \"Turning cooling fan on\"\
    nDescription: \
Turns cooling fan on\n\nCommand: \"FANCOOL OFF\"\nResponse: \"
    Turning cooling\
fan off\"\nDescription: Turns cooling fan off\n\nCommand: \"
    DISPLAY TEMP\"\n\
Response: \"<temperatures>\"\nDescription: returns inside and
    outside \
temperatures, and displays them on the box\n\nCommand: \"GREEN ON
    \"\nResponse:\
\"Green LED on\"\nDescription: Turns green LED on\n\nCommand: \"
    GREEN OFF\"\n\
Response: \"Green LED off\"\nDescription: Turns green LED off\n\
    nCommand: \"\
DISPLAY 8DIGIT\" after prompt \"<8 digit number>\"\nResponse: \"
    Enter 8 digit\
number to display:\"\nDescription: Prompts the user for a number
    to display on\
the 8 digit seven segment display.\n\nCommand: \"\
VERBOSE OFF\" \nDescription: Turns off ack responses to commands
    .\
the 8 digit seven segment display.\n\nCommand: \"\
VERBOSE ON\" \nDescription: Turns on ack responses to commands.\
the 8 digit seven segment display.\n\n\n";

float temp_outside;
int temp_inside;
char string_value[20];
char number_to_display[10];
String string_number;
int buff_byte;
int counter_display=0;
int int_to_display;
int read_loop;

switch(command_number){
  case -1:
    break;
  case 0:
    SerialUSB.print(help_text_var);
    break;
  case 1:
    SerialUSB.print(help_text_var);
    break;
  case 2:
```

```
314       if(verbose_mode == 1){
315         SerialUSB.print("Turning heater on\n");
316       }
317       digitalWrite(25, HIGH);   // HEAT ON, Red LED on, Blue LED
      off
318       break;                    // (gets shorted low by heater on)
319     case 3:
320       if(verbose_mode == 1){
321         SerialUSB.print("Turning heater off\n");
322       }
323       digitalWrite(25, LOW); // HEAT OFF
324       break;
325     case 4:
326       if(verbose_mode == 1){
327         SerialUSB.print("Turning heater fan on\n");
328       }
329       digitalWrite(23, HIGH); // FAN HEAT ON
330       break;
331     case 5:
332       if(verbose_mode == 1){
333         SerialUSB.print("Turning heater fan off\n");
334       }
335       digitalWrite(23, LOW); // FAN HEAT OFF
336       break;
337     case 6:
338       if(verbose_mode == 1){
339         SerialUSB.print("Turning cooling fan on\n");
340       }
341       digitalWrite(13, HIGH); // FAN COOL ON
342       break;
343     case 7:
344       if(verbose_mode == 1){
345         SerialUSB.print("Turning cooling fan off\n");
346       }
347       digitalWrite(13, LOW); // FAN COOL OFF
348       break;
349     case 8:
350       string_inside_temp(string_value);
351       SerialUSB.println(string_value);
352       break;
353     case 9:
354       string_outside_temp(string_value);
355       SerialUSB.println(string_value);
356       break;
357     case 10:
358       SerialUSB.print("Displaying temperatures in Celcius(outside
      temperature,...
359       inside temperature)\n");
360       displaytempertures();
361       break;
362     case 11:
363       if(verbose_mode == 1){
364         SerialUSB.print("Green LED on\n");
365       }
366       digitalWrite(30, HIGH); // LED ON GREEN
367       break;
368     case 12:
```

```
      if(verbose_mode == 1){
        SerialUSB.print("Green LED off\n");
      }
      digitalWrite(30, LOW); // LED OFF GREEN
      break;
    case 13:
      SerialUSB.print("Enter 8 digit number to display:\n");
      read_loop = 1;
      counter_display = 0;
      while(read_loop){
        buff_byte = SerialUSB.read();
        if(buff_byte>0){
          number_to_display[counter_display] = buff_byte;
          counter_display++;
          if((buff_byte == 13)||(buff_byte == 10)){
            number_to_display[counter_display] = 0 ;
            counter_display = 0;
            read_loop = 0;
          }
          delay(10);
        }
      }
      string_number = number_to_display;
      int_to_display = string_number.toInt();
      SerialUSB.print(int_to_display);
      displaynumber(int_to_display);
      break;
    case 14:
      if(verbose_mode==0){
        SerialUSB.print("Verbose mode already off\n");
      }else{
        SerialUSB.print("Verbose mode off\n");
      }
      verbose_mode = 0;
      break;
    case 15:
      if(verbose_mode==1){
        SerialUSB.print("Verbose mode already on\n");
      }else{
        SerialUSB.print("Verbose mode on\n");
      }
      verbose_mode = 1;
      break;
  }
  return;
}


#define BUFFER_SIZE 30
int byte_read;
int read_buffer[BUFFER_SIZE];
int read_counter=0;
int current_command = -1;

void loop() {
      byte_read = SerialUSB.read();
      if(byte_read>0){
```

```
426         read_buffer[read_counter] = byte_read;
427         read_counter++;
428         if((byte_read == 13)||(byte_read == 10)){
429           read_counter = 0;
430           int i = 0;
431           current_command = command_parse(read_buffer);
432         }
433       }
434       command_execute(current_command);
435       current_command = -1;
436       delay(10);
437 }
```

5 Appendix B: LabVIEW

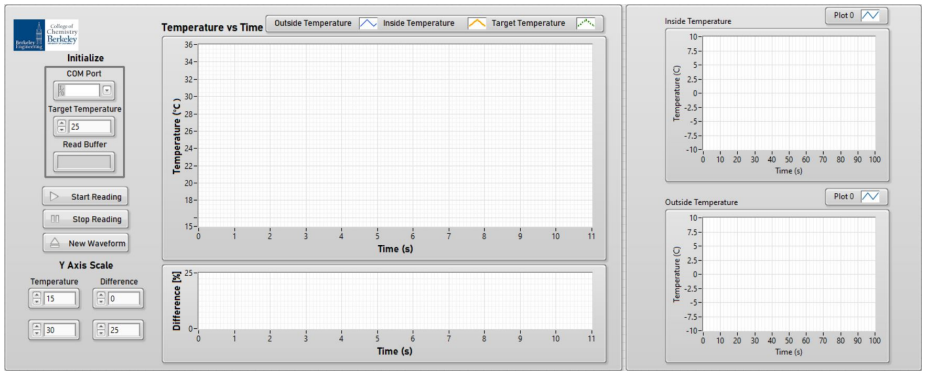


Figure 4: Front Panel on Program Open

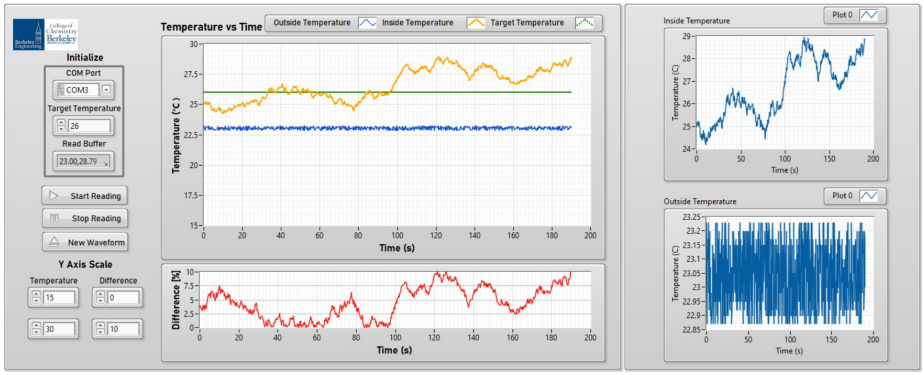


Figure 5: Front Panel Pure Reading, 200 Seconds

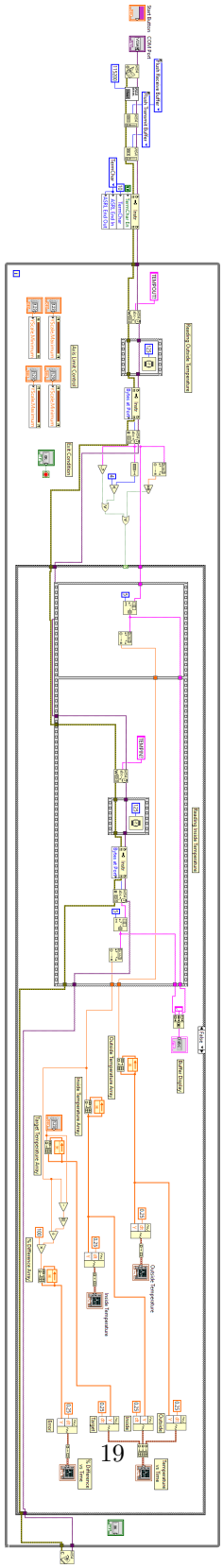


Figure 6: LabVIEW Block Diagram