2017-PHYS-339 Servo System

From McGill University Physics Department Technical Services Wiki
(Redirected from PHYS-339 Servo System)

Handout

- Main handout (http://www.ugrad.physics.mcgill.ca/resources/339/servo/servo.pdf)
- schmitt.ino (http://www.ugrad.physics.mcgill.ca/resources/339/servo/schmitt.ino) --- Arduino sketch

Calibration Data

- 2017-PHYS-339-Servo System Calibration Data

Software

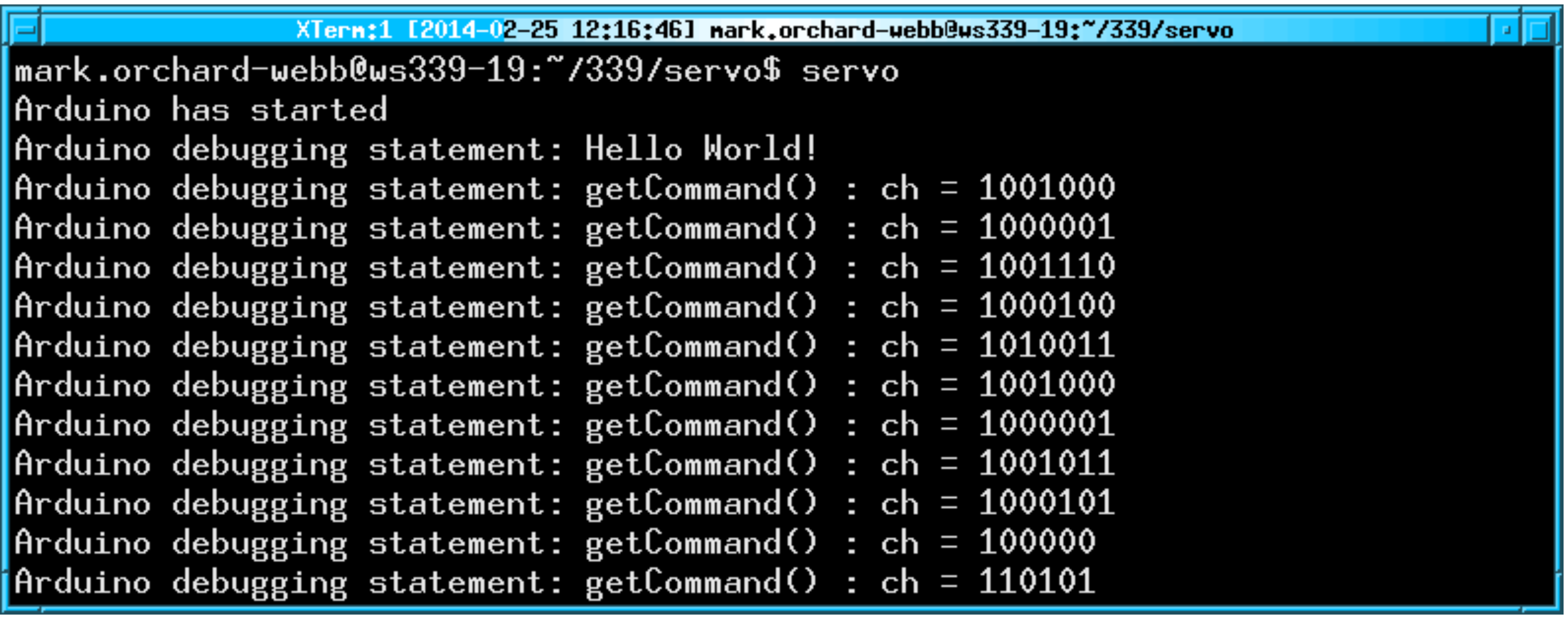
Introduction

The actual controller for this experiment is to be implemented as an Arduino sketch. The example, which implements a Schmitt controller can be found here (http://ganesh.ugrad.physics.mcgill.ca/resources/339/servo/schmitt.ino). There is also a GUI which interfaces with the arduino, allowing real-time monitoring and plotting of interesting variables, and saving into a MATLAB .mat file. I shall first give a walk through on the operation of the GUI. MATLAB you cry, what the what? Please see the discussion towards the bottom of the page.

Using GUI

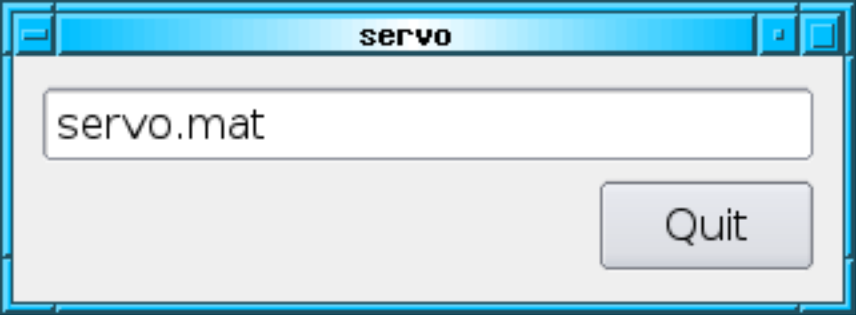
Note: in this year's code the output format has changed from MATLAB .mat file to Python/Numpy .npz.

The command to start the GUI is `servo`



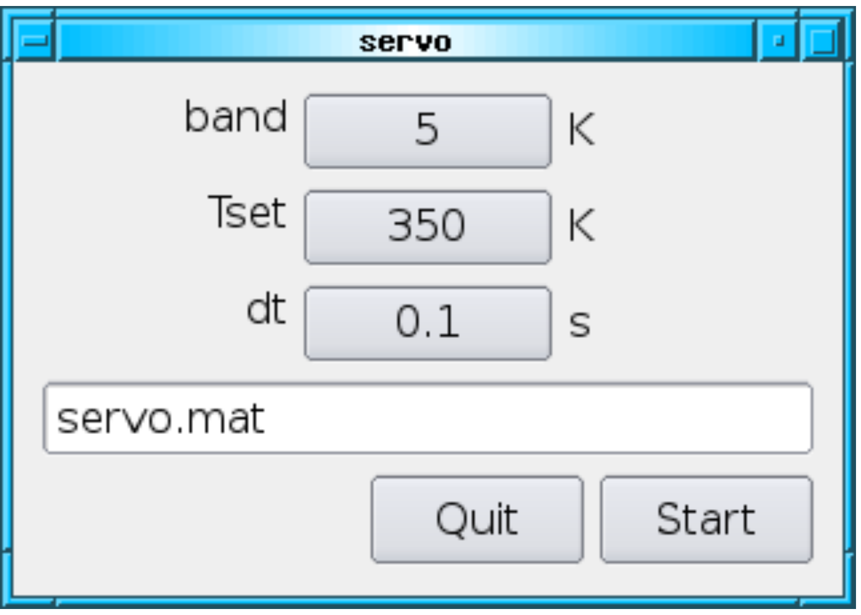
```
XTerm:1 [2014-02-25 12:16:46] mark.orchard-webb@ws339-19:~/339/servo
mark.orchard-webb@ws339-19:~/339/servo$ servo
Arduino has started
Arduino debugging statement: Hello World!
Arduino debugging statement: getCommand() : ch = 1001000
Arduino debugging statement: getCommand() : ch = 1000001
Arduino debugging statement: getCommand() : ch = 1001110
Arduino debugging statement: getCommand() : ch = 1000100
Arduino debugging statement: getCommand() : ch = 1010011
Arduino debugging statement: getCommand() : ch = 1001000
Arduino debugging statement: getCommand() : ch = 1000001
Arduino debugging statement: getCommand() : ch = 1001011
Arduino debugging statement: getCommand() : ch = 1000101
Arduino debugging statement: getCommand() : ch = 100000
Arduino debugging statement: getCommand() : ch = 110101
```

This should pop up the following window



servo.mat

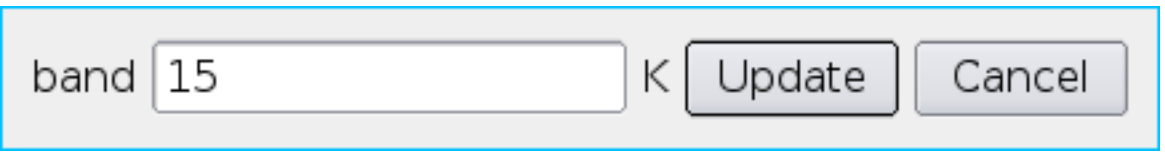
This should be quickly updated to



band 5 K
Tset 350 K
dt 0.1 s
servo.mat

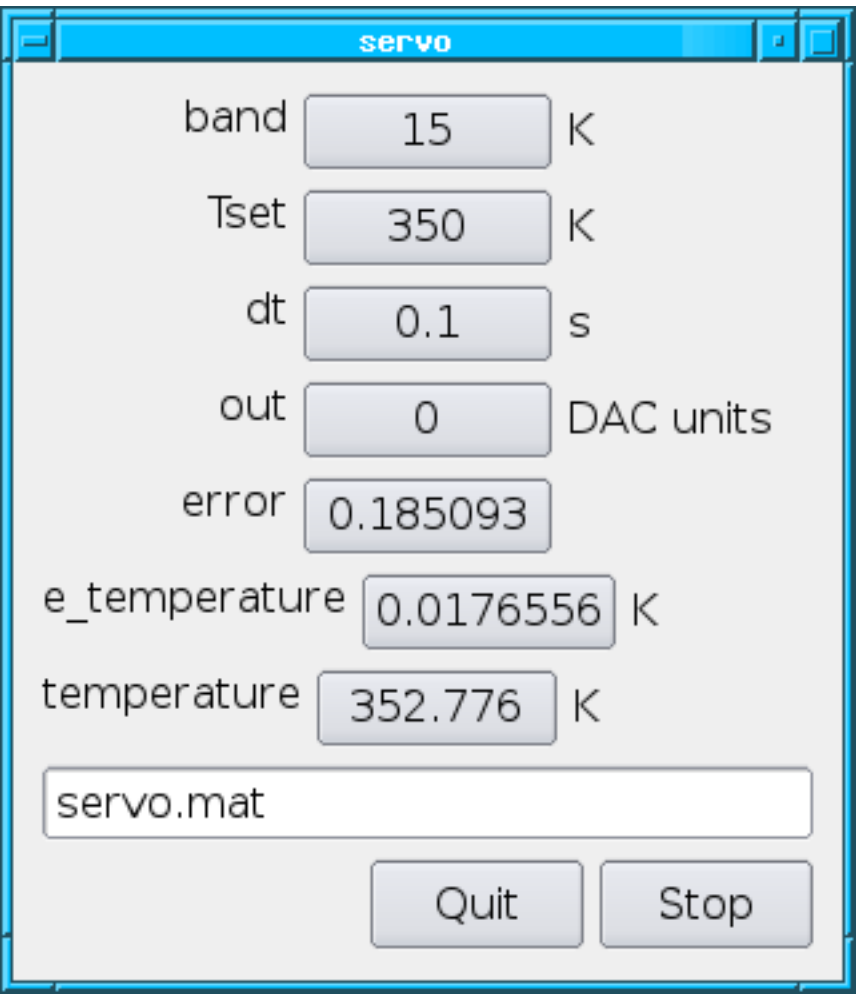
At this point the GUI has established contact with the Arduino and downloaded the variables with are to be initialized.

Clicking on one of the value buttons for these variables will pop up a dialog allowing values to be changed



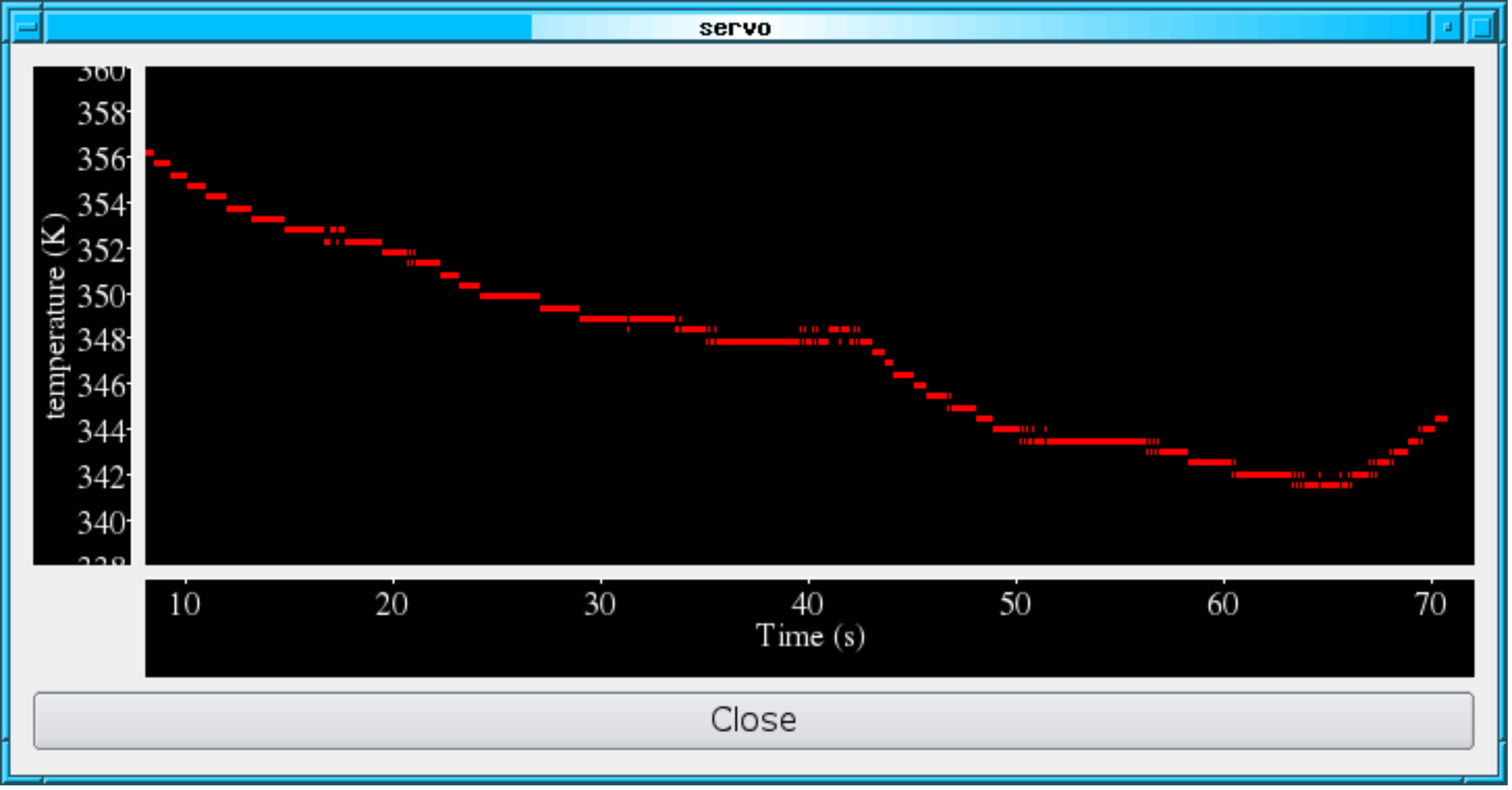
band 15 K

Clicking on the `Start` button will start the algorithm on the Arduino, and display the variables which are to be monitored.

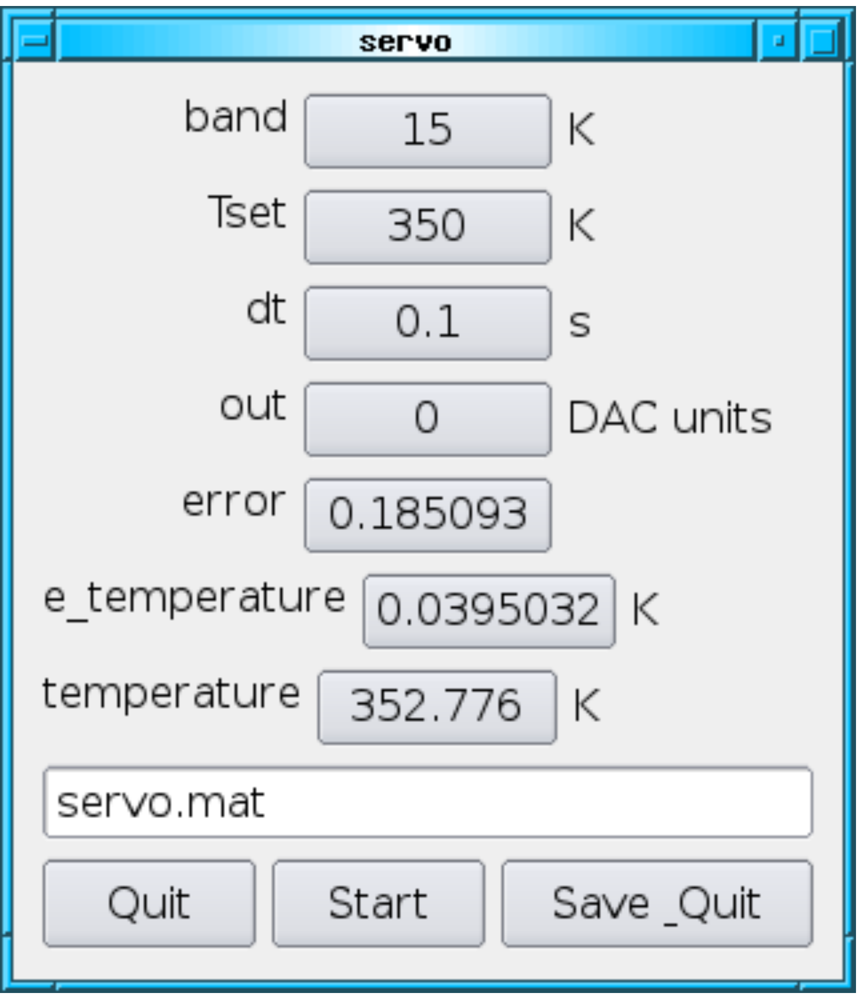


band 15 K
Tset 350 K
dt 0.1 s
out 0 DAC units
error 0.185093
e_temperature 0.0176556 K
temperature 352.776 K
servo.mat

The initialized variables can no longer be modified, but clicking on the real-time variables will pop up a graph of the variable.

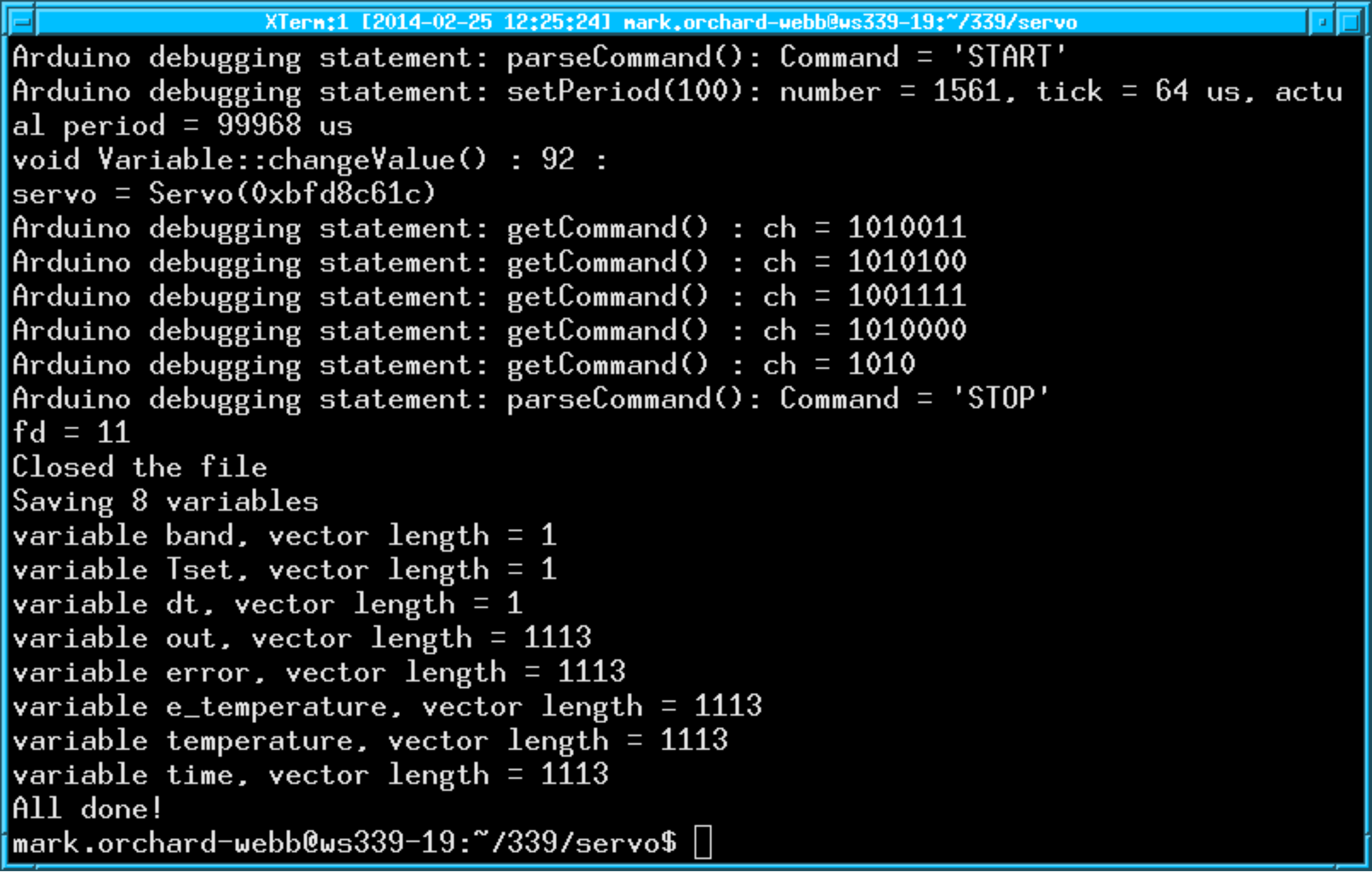


Clicking on stop will now reveal a new button `Save Quit`



band 15 K
Tset 350 K
dt 0.1 s
out 0 DAC units
error 0.185093
e_temperature 0.0395032 K
temperature 352.776 K
servo.mat

At this point you can again modify initialized variables, and click `Start`. If you click `Save Quit`, the following will happen



```
XTerm:1 [2014-02-25 12:25:24] mark.orchard-webb@ws339-19:~/339/servo
Arduino debugging statement: parseCommand(): Command = 'START'
Arduino debugging statement: setPeriod(100): number = 1561, tick = 64 us, actu
al period = 99968 us
void Variable::changeValue() : 92 :
servo = Servo(0xbfd8c61c)
Arduino debugging statement: getCommand() : ch = 1010011
Arduino debugging statement: getCommand() : ch = 1010100
Arduino debugging statement: getCommand() : ch = 1001111
Arduino debugging statement: getCommand() : ch = 1010000
Arduino debugging statement: getCommand() : ch = 1010
Arduino debugging statement: parseCommand(): Command = 'STOP'
fd = 11
Closed the file
Saving 8 variables
variable band, vector length = 1
variable Tset, vector length = 1
variable dt, vector length = 1
variable out, vector length = 1113
variable error, vector length = 1113
variable e_temperature, vector length = 1113
variable temperature, vector length = 1113
variable time, vector length = 1113
All done!
mark.orchard-webb@ws339-19:~/339/servo$ 
```

At this point all variables shown in the interface wave been saved into `servo.mat` which can be loaded into MATLAB.

The Arduino Sketch

I shall now discuss the relevant portions of the Arduino sketch

Declarations and macros

You can totally ignore this bit, but I have to put it before the bits you do want to work with.

```
void register_variable(double *address, const char *name, int record, const char *units);
#define RECORD(X,UNITS) register_variable(&X,#X, 1, UNITS)
#define INITIALIZE(X,VALUE,UNITS) X = VALUE; register_variable(&X,#X, 0, UNITS)
```

Global Variables

These are the variables which will be used by the algorithm, they are defined globally to make life easier. To those who feel terrible about this --- you have been brainwashed, it is a microcontroller! Any variable which you wish to appear in the GUI must appear here as a double.

```
/* These are the global variables used in the controller */

double dt;
double temperature;
double e_temperature;
double error;
double prevError;
double Tset;
double band;
double out;
```

userSetup

This is pretty much explained in the comments.

```
void userSetup() {
  /*
   * this function is called at program startup.  There are two useful things to do here
   *
   * RECORD(variableName,units)
   *   This will register a variable to be sent back to the computer at each time step
   *   The variable should be a globaly defined double precision variable (see temperature above)
   *   Units is a string used for labels
   *
   * INITIALIZE(variableName,value,units)
   *   This will register a variable which can be set from the computer
   *   The variable should be a globably defined double
   *   The value is used as the default value
   *   Units is a string used for labels
   */
  RECORD(temperature,"K");
  RECORD(e_temperature,"K");
  RECORD(error,"");
//  RECORD(derivative,"s<sup>-1</sup>"); /* you can use HTML, this will disp[ay a super-script */
  RECORD(out,"DAC units");
  INITIALIZE(dt,0.1,"s");
  INITIALIZE(Tset,350,"K");
  INITIALIZE(band,5,"K");
}
```

userAction

This is the exciting part. This function is called each time step, with happens every `dt` seconds. This is an implementation of a controller with hysteresis, it the temperature is below `Tset - band/2` (`Ton`) then maximum power is applied to the heater, if the temperature is above `Tset + band/2` (`Toff`) then zero power is applied. Between these limits nothing changes.

```
void userAction() {
  /*
   * This function will be called each time step
   */
  long sum = 0, sumsq = 0;
  int value;
  const int N = 16; /* number of samples measured per time step */
  for (int i = 0; i < N; i++) { /* record N samples */
    value = analogRead(0);
    sum += value; /* used to calculate mean */
    sumsq += value*(long)value; /* used to calulate variance, need to type cast to long because int is 16 bit */
  }
  double mu, sigma;
  mu = sum / (double) N;
  sigma = sqrt((sumsq - sum*mu)/N); /* expansion of definition of variance */
  const double ADCslope = 4.888e-3; /* 4.888mV / bit */
  const double ADCoffset = 1.02e-3; /* 1.02 mV */
  double vThermo, eVThermo;
  vThermo = ADCslope * mu + ADCoffset; /* convert means in ADC units to voltages */
  eVThermo = ADCslope * sigma;
  const double thermoSlope = 100; /* K/V */
  const double thermoOffset = 273; /* K */
  temperature = thermoSlope * vThermo + thermoOffset;
  e_temperature = eVThermo * vThermo;
  prevError = error;
  error = (temperature - Tset) / band;
  double Ton = Tset - band / 2;
  double Toff = Tset + band / 2;

  if ((out > 0) && (temperature > Toff)) {
    out = 0;
  } else if ((out == 0) && (temperature < Ton)) {
    out = 255;
  }
  int dacVal = out;
  /*
  Serial.print("DEBUG\tdacVal = ");
  Serial.print(dacVal);
  Serial.print("\n");
  */
  analogWrite(11, dacVal);
}
```

userStart

This function is called when the **Start** button if clicked in the GUI.

```
void userStart() {
/*
 * This function is called when the algorithm is started
 */
  out = 0;
}
```

userStop

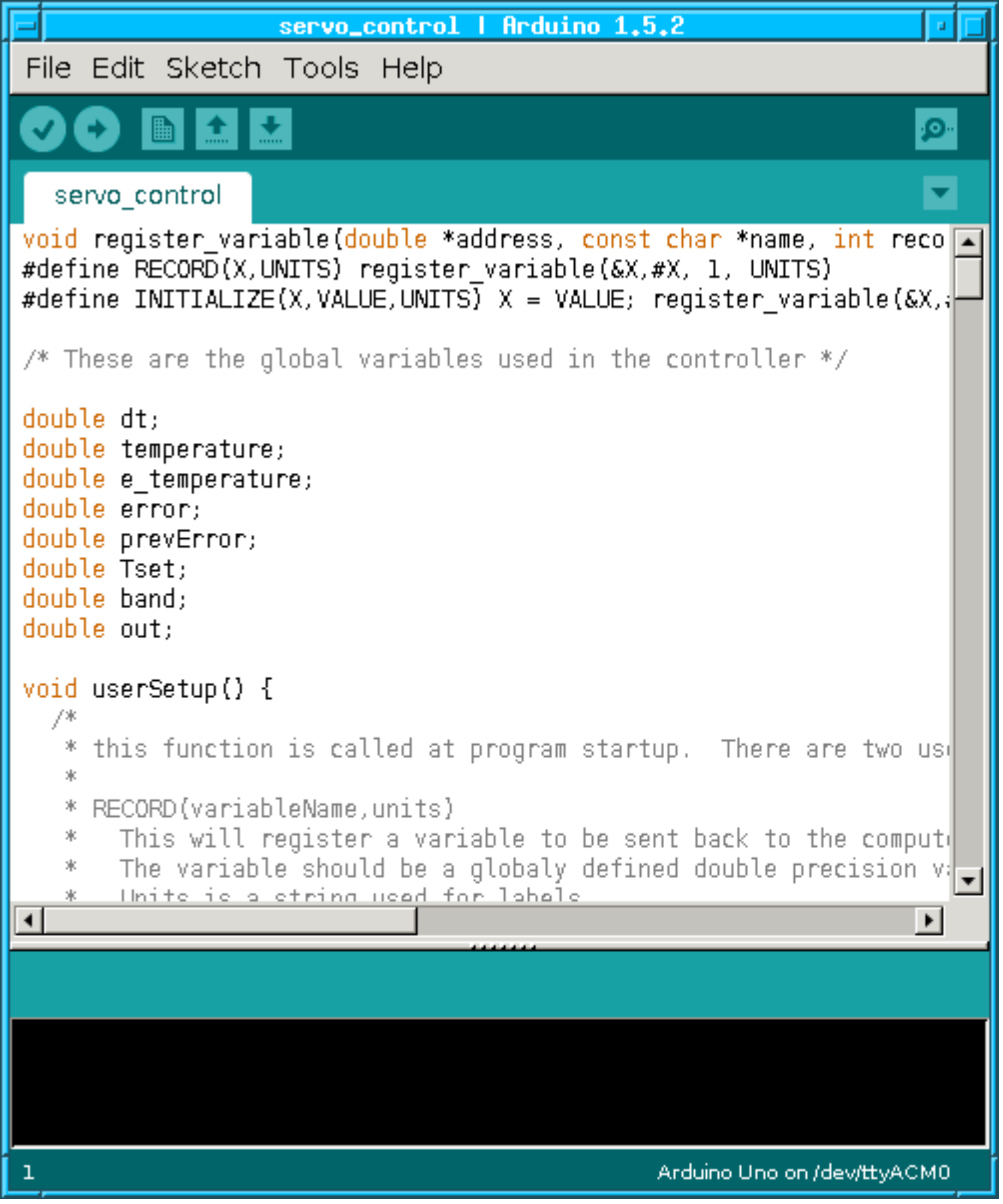
This function is called when the **Stop** button if clicked in the GUI.

```
void userStop() {
/*
 * This function is called when the algorithm is stopped
 * - Good place to turn off the power :)
 */
 analogWrite(11, 0); /* turn off power */
}
```

Programming the Arduino

The command to run the Arduino interface is, surprisingly enough, `arduino`. In addition to being written in Java, which is *a bad thing*, it has a few quirks. The most painful is the way it insists that the code, which is called a sketch, but is really C code, must be arranged in a folder with the same basename as the code, and this folder is squirelled away in a folder which is specified in the program preferences. For me, `schmitt` is stored as `Arduino/schmitt/schmitt.ino`. This will then appear in the list of sketchs under the `File->Sketchbook` menu.

To avoid having to mess around with this nonsense, I would recommend the following. Create a new sketch from the `File->New` menu. Delete everything in the new window. Select all the text in the web browser (link to sketch (http://ganesh.ugrad.physics.mcgill.ca/resources/339/servo/schmitt.ino))(`control-A`, copy it into the copy buffer (either right click->copy or `control-C`), then paste into the new window, then name it using `File->Save As`.


servo_control | Arduino 1.5.2
File Edit Sketch Tools Help
servo_control
1
Arduino Uno on /dev/ttyACM0

```
void register_variable(double *address, const char *name, int reco
#define RECORD(X,UNITS) register_variable(&X,#X, 1, UNITS)
#define INITIALIZE(X,VALUE,UNITS) X = VALUE; register_variable(&X,

/* These are the global variables used in the controller */

double dt;
double temperature;
double e_temperature;
double error;
double prevError;
double Tset;
double band;
double out;

void userSetup() {
  /*
   * this function is called at program startup.  There are two us
   *
   * RECORD(variableName,units)
   *   This will register a variable to be sent back to the comput
   *   The variable should be a globaly defined double precision v
   *   Units is a string used for labels
```

Before attempting to access the Arduino, you need to make sure the serial port is set correctly. Click the `Tools->Serial Port` menu. The Arduino port will show up as `ttyACMn`, where n is a small integer.

To upload the sketch to the Arduinio, click the right arrow in a circle icon, or type `control-U`. This will compile the code and if successful upload the binary to the Arduino.

Reading the data in Python

I debated for a while about the relative merits of replacing the code which saves the recorded data in a format native to Python, but it was evident that it was going to be messy. The are an arbitrary number of both named variables and named vectors. It would be possible to generate a structure, but that would be pretty ugly and would hard code a variable name. Could get around that by creating a class and initializing the structure from a file, but what format for the file. Apparently Python nerds save structures using a pickle library. Right, so you save data by pickling it. Go outside nerds, you are breaking your mother's hearts. Ultimately the solution is a variant of the second approach. There is subset of the SciPy library which is dedicated to manipulating MATLAB .mat file. This is extremely satisfying because I don't have to break my code which creates the .mat file --- which I did derive way too much satisfaction from and I don't have to take any responsibility for the ugliness of the resulting Python data structure.

An example of this usage is shown in plotservo.py

```
import matplotlib.pyplot as plt
import scipy.io as sio

data = sio.loadmat('servo.mat')

plt.plot(data['time'][0],data['temperature'][0])
plt.show()
```

Historical Links

- 2007-PHYS-339 Servo System
- 2006-PHYS-339 Servo System
- 2008-PHYS-339 Servo System
- 2009-PHYS-339 Servo System
- 2012-PHYS-339 Servo System
- 2013-PHYS-339 Servo System
- 2014-PHYS-339 Servo System
- 2015-PHYS-339 Servo System
- 2016-PHYS-339 Servo System

2017-PHYS-339-Servo System Calibration Data

From McGill University Physics Department Technical Services Wiki

Group 1

- Heater resistance: 19.6 Ω
- Max current: 0.548
- Thermocouple calibration: 272.2 Kelvin + 100 Kelvin/Volt V_{th}

Group 2

- Heater resistance: 14.0 Ω
- Max current: 0.532
- Thermocouple calibration: 272.9 Kelvin + 100 Kelvin/Volt V_{th}

Group 3

- Heater resistance: 19.9 Ω
- Max current: 0.535
- Thermocouple calibration: 272.2 Kelvin + 100 Kelvin/Volt V_{th}

Group 4

- Heater resistance: 19.6 Ω
- Max current: 0.531
- Thermocouple calibration: 272.4 Kelvin + 100 Kelvin/Volt V_{th}

Group 5

- Heater resistance: 19.3 Ω
- Max current: 0.534
- Thermocouple calibration: 269.2 Kelvin + 100 Kelvin/Volt V_{th}

Group 6

- Heater resistance: 17.0 Ω
- Max current: 0.531
- Thermocouple calibration: 273.7 Kelvin + 100 Kelvin/Volt V_{th}

Group 7

- Heater resistance: 20.1 Ω
- Max current: 0.552
- Thermocouple calibration: 275.0 Kelvin + 100 Kelvin/Volt V_{th}

Group 9

- Heater resistance: 19.9 Ω
- Max current: 0.546
- Thermocouple calibration: 269.9 Kelvin + 100 Kelvin/Volt V_{th}

Group 11

- Heater resistance: 17.6 Ω
- Max current: 0.548
- Thermocouple calibration: 273.3 Kelvin + 100 Kelvin/Volt V_{th}

Group 12

- Heater resistance: 19.2 Ω
- Max current: 0.529
- Thermocouple calibration: 271.3 Kelvin + 100 Kelvin/Volt V_{th}

Group 13

- Heater resistance: 24.6 Ω
- Max current: 0.549
- Thermocouple calibration: 271.1 Kelvin + 100 Kelvin/Volt V_{th}

Group 14

- Heater resistance: 22.6 Ω
- Max current: 0.512
- Thermocouple calibration: 272.3 Kelvin + 100 Kelvin/Volt V_{th}

Group 18

- Heater resistance: 21.7 Ω
- Max current: 0.542
- Thermocouple calibration: 274.1 Kelvin + 100 Kelvin/Volt V_{th}

Group 19

- Heater resistance: 20.6 Ω
- Max current: 0.531
- Thermocouple calibration: 273.9 Kelvin + 100 Kelvin/Volt V_{th}