APPENDIX F

Microcomputer Control for AASHTO T 161 "A" F–T Cabinet

The standard AASHTO T 161 "A" freeze–thaw (F-T) cabinet is a relatively simple and straightforward design. The insulated cabinet contains a freezing table on which stainless steel pans sit filled with concrete specimens in water. Freezing occurs from the bottom up, and heating is provided by resistive heaters inserted between the sample pans. The freezing and heating capabilities are designed for the rapid "A" version of 4–5 cycles per day. The temperature is controlled from the center of a dummy concrete beam located near the middle of the test samples.

In the early units, a mechanical temperature switch transferred power between the cooling unit and heating units when the upper or lower set limit was reached. Continuous observation was often required as the set limits needed adjusting while the cabinet came to equilibrium from ambient temperature, or inevitably, ice or condensation began accumulating inside the cabinet. A chart recorder wheel was installed as secondary verification as the mechanical switches were often only triggered near the desired settings.

As computer controls became available, proprietary computer controls became an option and then became the standard. The computer-controlled systems were all designed only to perform AASHTO T 161 "A" without any flexibility for other test methods.

In any arrangement, the AASHTO T 161 "A" cabinet has a temperature sensor that controls a switch to turn on/off the resistive heaters or cooling unit. The Raspberry Pi control used in this study employed a Campbell Scientific T107 thermistor connected to an analog to digital (A/D) shield. Low voltage AC/DC relays were controlled by the Raspberry Pi to initiate either heating or cooling.

Table F-1 provides the parts list required to convert either the fully mechanical or semi-computerized cabinets to Raspberry Pi control.

Table F-1. Parts list for Raspberry Pi control.

Item	Manufacturer	Description	Part Name/Number	Quantity	Piece/ Each	Price
1	Omron	Solid state relay - industrial mount 5-24VDC/240VAC 25A	G3NA-225B DC5-24	2	$31.92	$64
2	Bussmann	Fuse holder	BM6031PQ for 15A BAF type fuse	2	$11	$22
3	Bussmann	Fuse holder	BM6032PQ for 15A CC type fuse	1	$14	$14
4	Schneider Electric	Single contactor	Square D Definite Purpose Contactor-single pole	1	$40	$40
5	Schneider Electric	Double contactor	Square D/8910DP32V02/ Contactor, 600 VAC, 2 pole	1	$46	$46
6	Bussmann	15A CC time delay fuse	P-CC15 Amp, 600V, Class CC, LP-CC-15, LP-CC15	3	$8	$24
7	Bussmann	20A CC time delay fuse	20A CC time delay fuse	1	$8	$8
8	Grainger	Terminal strip	15-pole electric terminal block	1	$15	$15
9	Raspberry Pi	Microcomputer	Raspberry Pi 3 Model B	1	$35	$35
10	Waveshare	Raspberry Pi high-precision AD/DA board	RB-Wav-08	1	$28	$28
11	Campbell Scientific	Temperature sensor	Tl07	1	$125	$125
Total for All						**$420**

A fully computerized version of the cabinet will include the necessary contactors, fusing, and terminal blocks. With a computerized version, the only parts required are the lower voltage relays, Raspberry Pi, AD/DA board, and T107 thermistor.

Figure F-1 shows the wiring schematic for computerizing the F-T cabinet.

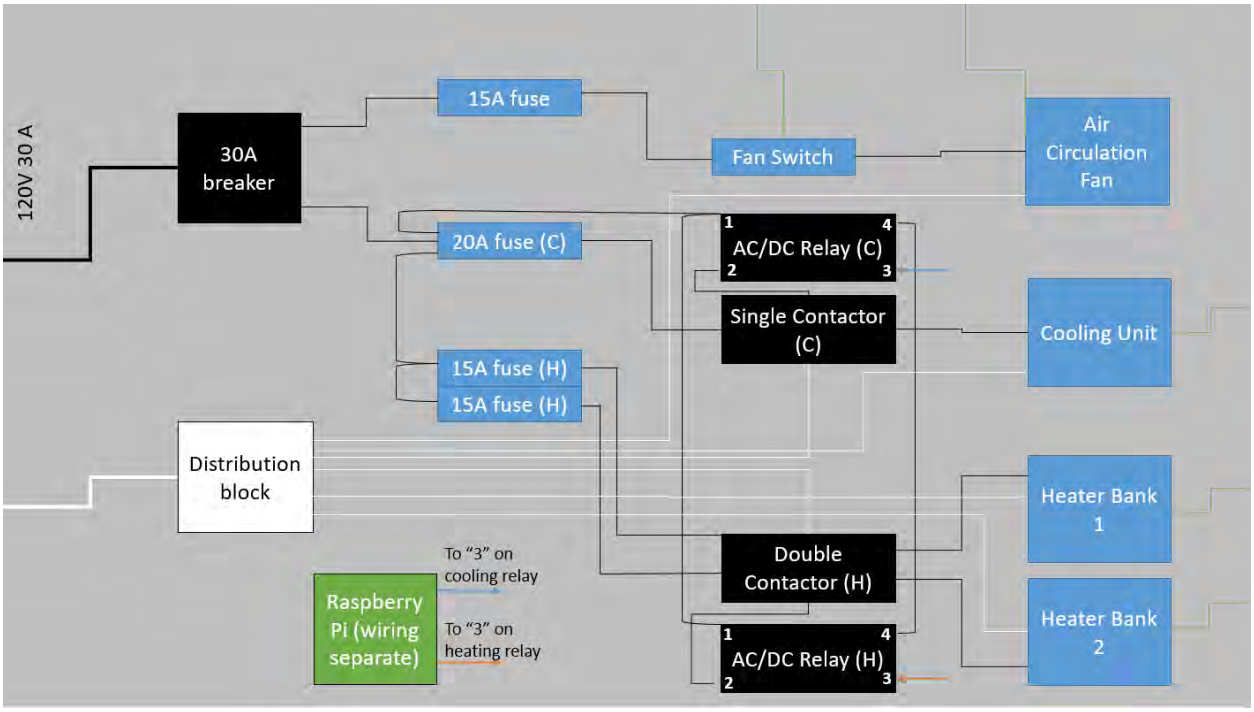


Figure F-1. Wiring diagram for Raspberry Pi control.

Wire gauges should be at least 10 ga. from the wall outlet to the main breaker. All AC wires should be 12 ga. Ensure the unit is disconnected from the wall before any wiring or rewiring. Wiring should only be performed by someone familiar with electrical systems. Figure F-2 shows the wiring setup on a prototype board.

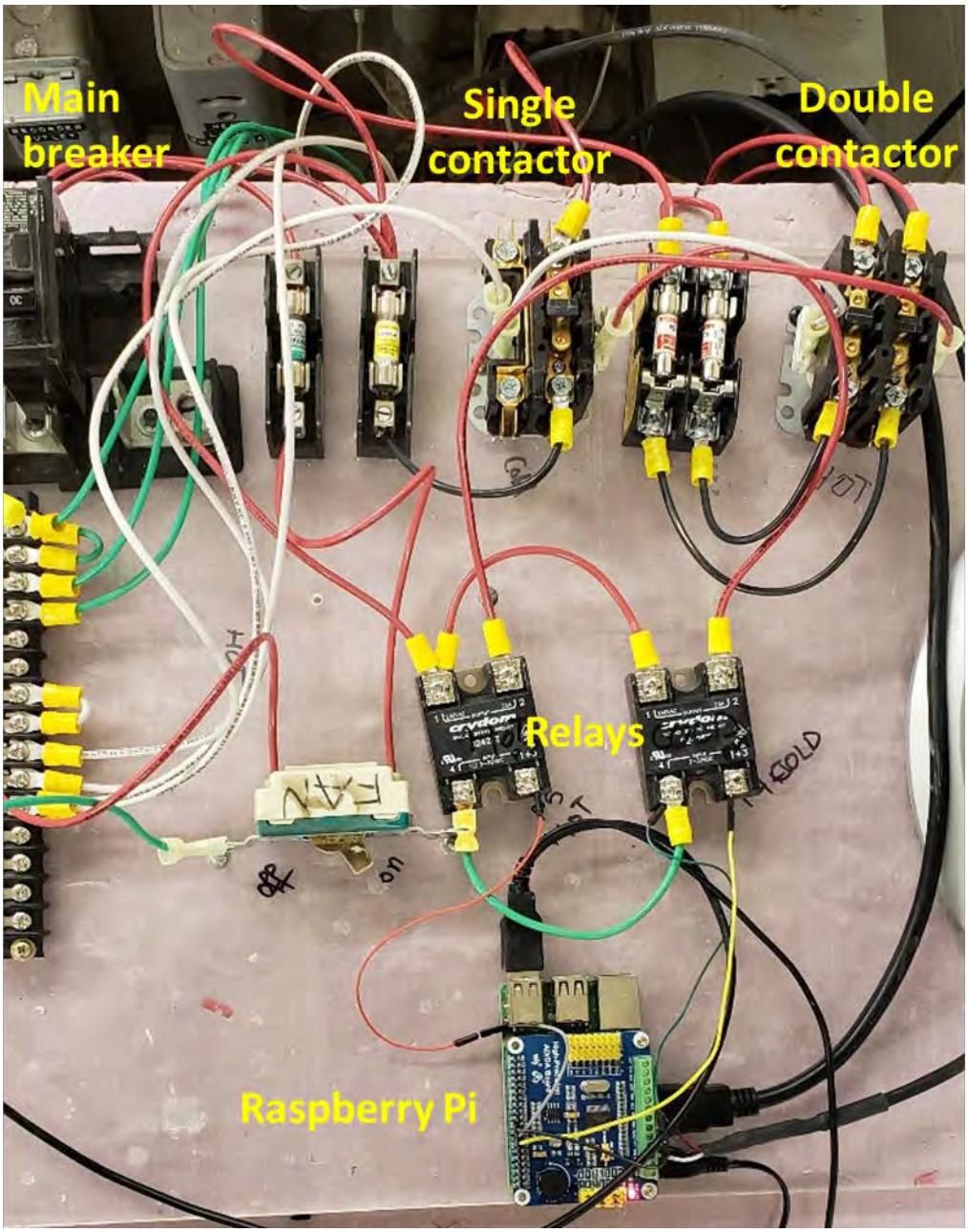


Figure F-2. Expanded wiring overview.

Figure F-3 shows the overview of the Raspberry Pi with the AD/DA shield attached.

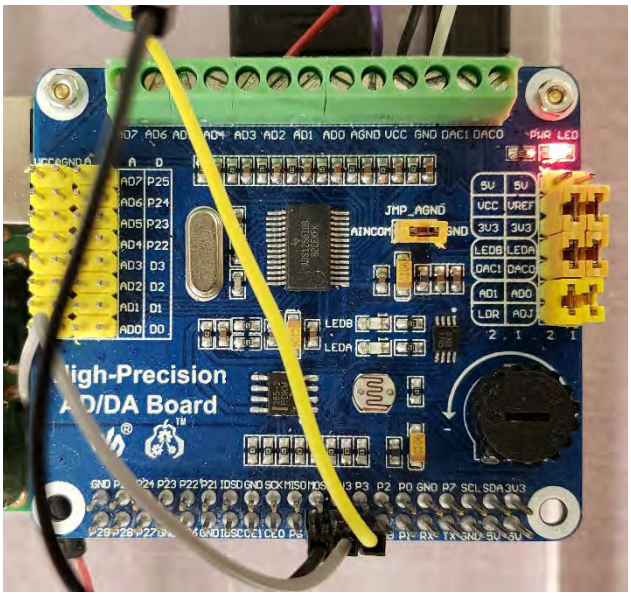
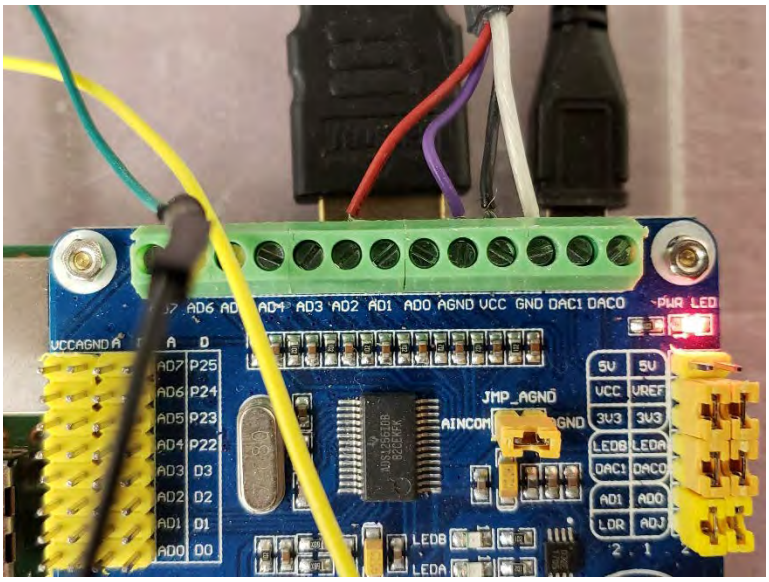


Figure F-3. Overview of AD/DA board wiring.

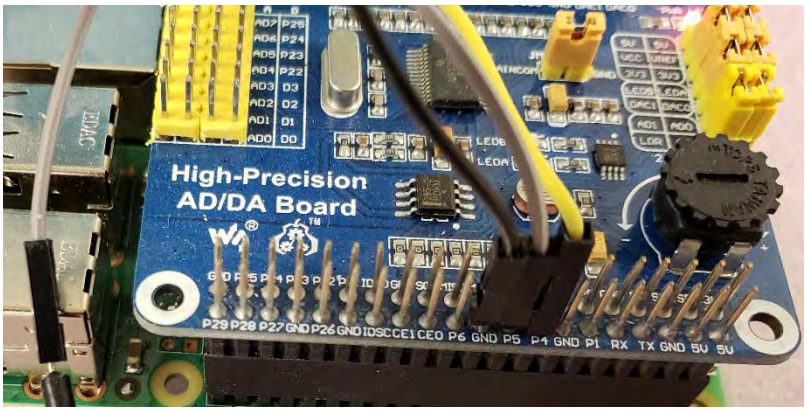
The Campbell Scientific T107 temperature sensor is wired into the top (analog) side and into the relay controls in the bottom (digital) side. Figure F-4 shows the wiring from the temperature sensor in detail.



Note: Red = AD2, Purple = AGND, Black = VCC, White = GND.

Figure F-4. Analog connection for temperature sensor.

The 3.3 v output from the sensor is coded into the AD2 terminal with the purple ground into AGND, black into VCC, and white into GND. The digital output is shown in Figure F-5 where the ground is connected to 4 on the relays, the P5 post is connected to the 3 control on the heater relay, and P4 connected to the 3 on the cooling relay.



Note: GND = Relay 4, P5 = Heater Relay 3, P4 = Cooling Relay 3.

Figure F-5. Digital connection for relay control.

Raspberry Pi Program for AASHTO T 161 “A”

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import time, math
import sys
from ADS1256_definitions import *
from pipyadc import ADS1256
import RPi.GPIO as GPIO
import os



# Input pin for the potentiometer on the Waveshare Precision ADC
board:
POTI = POS_AIN0|NEG_AINCOM
# Light dependant resistor of the same board:
LDR  = POS_AIN1|NEG_AINCOM
# The other external input screw terminals of the Waveshare
board:
EXT2, EXT3, EXT4 = POS_AIN2|NEG_AINCOM, POS_AIN3|NEG_AINCOM,
POS_AIN4|NEG_AINCOM
EXT5, EXT6, EXT7 = POS_AIN5|NEG_AINCOM, POS_AIN6|NEG_AINCOM,
POS_AIN7|NEG_AINCOM
POTI_INVERTED = POS_AINCOM|NEG_AIN0
SHORT_CIRCUIT = POS_AIN0|NEG_AIN0
CH_SEQUENCE = (POTI, LDR, EXT2, EXT3, EXT4, EXT7, POTI_INVERTED,
SHORT_CIRCUIT)

# Campbell polynomial fit data for T107 temperature probe
Temp_Coeff = [-53.4601, 90.807, -83.257, 52.283, -16.723, 2.211]
#Temp_Coeff = [-75.499, 0.60781, -0.0011973, 1.0489e-6, 0, 0]
### Initialise ADC object:
ads = ADS1256()
### Gain and offset self-calibration:
ads.cal_self()

########################################
########################################
########################################
# Cycle Parameters
N = 30# Number of cycles
Tmax = 37 #Upper temp limit in F
Tmin = 3 # Lower temp limit in F
########################################
########################################
########################################
```

```
# Set GPIO to Broadcom numbering system
GPIO.setmode(GPIO.BOARD)
#GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Setting up the two relay pins
Heat_Pin = 18 # This is the PIN number not the GPIOXX number
Cool_Pin = 16 #GPIO24, 16=GPIO23
GPIO.setup(Heat_Pin, GPIO.OUT)
GPIO.setup(Cool_Pin, GPIO.OUT)
GPIO.output(Heat_Pin, False)
GPIO.output(Cool_Pin, False)
heat_status = 0
cold_status = 0

# Correction factor to match known thermometer values
adjustment_value = 1.0

# Setting up the logging file, stores in Log_Files directory
gg = 0
while os.path.exists("Log_Files/datalog%s.csv" % gg):
     gg+=1
header_string = "Time, Temp (F), Heat, Cool, Cycle Num,
Voltage/reading \n"
fh = open("Log_Files/datalog%s.csv" % gg,"a")
fh.write(header_string)
fh.close()

DEBUG_MODE = False


def temperature_reading():
     temp_volt = 0
     for g in range(0,25):
          raw_channels = ads.read_oneshot(EXT2)
          voltages     = raw_channels * ads.v_per_digit
          temp_volt = temp_volt + voltages
     temp_volt = temp_volt/25 # averaging 100 samples to get
better voltage estimate
     t107_volt = temp_volt/4.388*800
     t107_temp = Temp_Coeff[0] + Temp_Coeff[1]*t107_volt +
Temp_Coeff[2]*t107_volt**2 + Temp_Coeff[3]*t107_volt**3 +
Temp_Coeff[4]*t107_volt**4 + Temp_Coeff[5]*t107_volt**5
     t107_temp = 9.0/5.0*t107_temp + 32.0 # Converting Celsius
to F
     #print '{:.1f}'.format(t107_temp) + ' F'
```

```
     return t107_temp, t107_volt # returning both in case a
post-test calibration is needed



initial_time = time.time()
if (DEBUG_MODE == True): # Debug mode enables turning relays on
manually
     while True: # Run debug code forever
          print "Which relay do you want to turn on? Relay will
be on for 5 seconds"
          output_status = input("0 for Heat, 1 for Cool, Ctrl+C
to quit ")
          #output_status = 3

          if (output_status == 0):
               active_pin = Heat_Pin
               device = "Heat"
               heat_status = 1
               cold_status = 0
          if (output_status == 1):
               active_pin = Cool_Pin
               device = "Cool"
               heat_status = 0
               cold_status = 1
          GPIO.output(active_pin, True)
          #time.sleep(20.0)
          #if (output_status != 1 and output_status != 0):
          #    device = "None"
          #    print "Error, incorrect choice"
          #    break
          temp_temp = 0
          for i in range(0,30):

               [temp, volt] = temperature_reading()
               #print temp, volt
          #    GPIO.output(active_pin, True)
               print '{:.1f}'.format(temp) + ' F'
               #device = "None"
               #print "Temp: ", '{:.1f}'.format(temp), device, "
is ON"
               fh = open("Log_Files/datalog%s.csv" % gg,"a")
               log_data = '{:.1f}'.format(time.clock()) + ',' +
'{:.1f}'.format(temp) + ',' + str(heat_status) + ',' +
str(cold_status) + '\n'
               fh.write(log_data)
               fh.close()
```

```
                time.sleep(0.5)
          GPIO.output(Heat_Pin, False)
          GPIO.output(Cool_Pin, False)
else:
     prev_time = time.time()
     for i in range(0,N):
          [temp, volt] = temperature_reading()
          while temp > Tmin:
               if (time.time() - prev_time >= 5.0):
                    prev_time = time.time()
                    GPIO.output(Cool_Pin, True)
                    cold_status = 1
                    heat_status = 0
                    [temp, volt] = temperature_reading()
                    print '{:.1f}'.format(time.time()-
initial_time),"Temp: ", '{:.1f}'.format(temp), "F, COLD is ON,
Cycles=", i, '{:.4f}'.format(volt)
                    fh = open("Log_Files/datalog%s.csv" %
gg,"a")
                    log_data = '{:.1f}'.format(time.time()-
initial_time) + ',' + '{:.1f}'.format(temp) + ',' +
str(heat_status) + ',' + str(cold_status) + ',' +str(i) + ','  +
'{:.4f}'.format(volt) + '\n'
                    fh.write(log_data)
                    fh.close()

          [temp, volt] = temperature_reading()
          GPIO.output(Cool_Pin, False)
          while temp < Tmax:
               if (time.time() - prev_time >= 5.0):
                    prev_time = time.time()
                    GPIO.output(Heat_Pin, True)
                    cold_status = 0
                    heat_status = 1
                    [temp, volt] = temperature_reading()
                    print '{:.1f}'.format(time.time()-
initial_time), "Temp: ", '{:.1f}'.format(temp), "F, HEAT is ON,
Cycles=", i, '{:.4f}'.format(volt)
                    fh = open("Log_Files/datalog%s.csv" %
gg,"a")
                    log_data = '{:.1f}'.format(time.time()-
initial_time) + ',' + '{:.1f}'.format(temp) + ',' +
str(heat_status) + ',' + str(cold_status) + ',' +str(i) +  ',' +
'{:.4f}'.format(volt) +  '\n'
                    fh.write(log_data)
                    fh.close()
```

```
     GPIO.output(Heat_Pin, False)

# End of F-T test, turn off all systems #
GPIO.output(Heat_Pin,False)
GPIO.output(Cool_Pin,False)
```

Raspberry Pi Program for CDF Ramping Program

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import time, math
import sys
from ADS1256_definitions import *
from pipyadc import ADS1256
import RPi.GPIO as GPIO
import os

# Input pin for the potentiometer on the Waveshare Precision ADC
board:
POTI = POS_AIN0|NEG_AINCOM
# Light dependant resistor of the same board:
LDR  = POS_AIN1|NEG_AINCOM
# The other external input screw terminals of the Waveshare
board:
EXT2, EXT3, EXT4 = POS_AIN2|NEG_AINCOM, POS_AIN3|NEG_AINCOM,
POS_AIN4|NEG_AINCOM
EXT5, EXT6, EXT7 = POS_AIN5|NEG_AINCOM, POS_AIN6|NEG_AINCOM,
POS_AIN7|NEG_AINCOM
POTI_INVERTED = POS_AINCOM|NEG_AIN0
SHORT_CIRCUIT = POS_AIN0|NEG_AIN0
CH_SEQUENCE = (POTI, LDR, EXT2, EXT3, EXT4, EXT7, POTI_INVERTED,
SHORT_CIRCUIT)

# Campbell polynomial fit data for T107 temperature probe
Temp_Coeff = [-53.4601, 90.807, -83.257, 52.283, -16.723, 2.211]
#Temp_Coeff = [-75.499, 0.60781, -0.0011973, 1.0489e-6, 0, 0]
### Initialise ADC object:
ads = ADS1256()
### Gain and offset self-calibration:
ads.cal_self()

########################################
########################################
########################################
# Cycle Parameters
N = 4# Number of cycles
Hot_Thresh = 2 # Hot side +/- trigger threshold
Cold_Thresh = 2 # Cold side +/- trigger threshold
Ramp_Thresh = 2 # Trigger threshold during ramp (same for
heating and cooling)
########################################
########################################
########################################
```

```
# Bottom and top temperatures
Tmax = 68.0# Upper temp limit in F
Tmin = -4.0# Lower temp limit in F

# Set GPIO to Broadcom numbering system
GPIO.setmode(GPIO.BOARD)
#GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

# Setting up the two relay pins
Heat_Pin = 18 # This is the PIN number not the GPIOXX number
Cool_Pin = 16 #GPIO24, 16=GPIO23
GPIO.setup(Heat_Pin, GPIO.OUT)
GPIO.setup(Cool_Pin, GPIO.OUT)
GPIO.output(Heat_Pin, False)
GPIO.output(Cool_Pin, False)
heat_status = 0
cold_status = 0

# Correction factor to match known thermometer values
adjustment_value = 1.0

# Setting up the logging file, stores in Log_Files directory
gg = 0
while os.path.exists("Log_Files/datalog%s.csv" % gg):
     gg+=1
header_string = "Time, Temp (F), Heat, Cool, Cycle Num,
Voltage/reading \n"
fh = open("Log_Files/datalog%s.csv" % gg,"a")
fh.write(header_string)
fh.close()

DEBUG_MODE = False


def temperature_reading():
     temp_volt = 0
     for g in range(0,25):
          raw_channels = ads.read_oneshot(EXT2)
          voltages     = raw_channels * ads.v_per_digit
          temp_volt = temp_volt + voltages
     temp_volt = temp_volt/25 # averaging 100 samples to get
better voltage estimate
     t107_volt = temp_volt/4.388*800
```

```
     t107_temp = Temp_Coeff[0] + Temp_Coeff[1]*t107_volt +
Temp_Coeff[2]*t107_volt**2 + Temp_Coeff[3]*t107_volt**3 +
Temp_Coeff[4]*t107_volt**4 + Temp_Coeff[5]*t107_volt**5
     t107_temp = 9.0/5.0*t107_temp + 32.0 # Converting Celsius
to F
     #print '{:.1f}'.format(t107_temp) + ' F'
     return t107_temp, t107_volt # returning both in case a
post-test calibration is needed



initial_time = time.time()
if (DEBUG_MODE == True): # Debug mode enables turning relays on
manually
     while True: # Run debug code forever
          print "Which relay do you want to turn on? Relay will
be on for 5 seconds"
          output_status = input("0 for Heat, 1 for Cool, Ctrl+C
to quit ")
          #output_status = 3

          if (output_status == 0):
               active_pin = Heat_Pin
               device = "Heat"
               heat_status = 1
               cold_status = 0
          if (output_status == 1):
               active_pin = Cool_Pin
               device = "Cool"
               heat_status = 0
               cold_status = 1
          GPIO.output(active_pin, True)
          #time.sleep(20.0)
          #if (output_status != 1 and output_status != 0):
          #    device = "None"
          #    print "Error, incorrect choice"
          #    break
          temp_temp = 0
          for i in range(0,30):

               [temp, volt] = temperature_reading()
               #print temp, volt
          #    GPIO.output(active_pin, True)
               print '{:.1f}'.format(temp) + ' F'
               #device = "None"
               #print "Temp: ", '{:.1f}'.format(temp), device, "
is ON"
```

```
                fh = open("Log_Files/datalog%s.csv" % gg,"a")
                log_data = '{:.1f}'.format(time.clock()) + ',' +
'{:.1f}'.format(temp) + ',' + str(heat_status) + ',' +
str(cold_status) + '\n'
                fh.write(log_data)
                fh.close()
                time.sleep(0.5)
          GPIO.output(Heat_Pin, False)
          GPIO.output(Cool_Pin, False)
else:
     prev_time = time.time()
     [temp, volt] = temperature_reading()
     # Checking if tank is at correct temperature before
starting the cycle process #
     while ((temp < Tmin-Cold_Thresh) or (temp >
Tmax+Hot_Thresh)): # and (time.time() - prev_time >= 5.0)):
          [temp, volt] = temperature_reading()
          if (time.time() - prev_time >= 5.0):
               if (temp > Tmax+Hot_Thresh):
                    GPIO.output(Heat_Pin,False)
                    GPIO.output(Cool_Pin,True)
                    prev_time = time.time()
                    print 'PREP PHASE: Temp is ',
'{:.1f}'.format(temp), ' and Cool is ON'
               elif (temp < Tmin-Cold_Thresh):
                    GPIO.output(Cool_Pin,False)
                    GPIO.output(Heat_Pin,True)
                    prev_time = time.time()
                    print 'PREP PHASE: Temp is ',
'{:.1f}'.format(temp), ' and Heat is ON'

#######--------#########-------###########---------
###########
     print 'PREP COMPLETE: Beginning Ramp Test with ', N, '
cycles'
     start_time = time.time() # Grabbing cycle start time for
logging purposes (will subtract this off to make times relative
to start time)
     for i in range(0,N):
          [temp, volt] = temperature_reading()
          # 4 hour cycle from 68 to -4
          wait_time = time.time() # initializing timer for 4
hour ramp
          while (time.time() < wait_time + 14400.0): #4 hour
ramp
               if (time.time() - prev_time) >= 60.0: # 60 sec
update rate
```

```
                    cycle_time = time.time() - start_time #this
gives time since start of cycle
                    desired_T = Tmax - ((Tmax -
Tmin)*cycle_time*0.25/3600.0) # desired temp at current time
                    [temp, volt] = temperature_reading()
                    if (temp > desired_T + Ramp_Thresh): # Too
high above threshold, need to cool
                         GPIO.output(Heat_Pin,False)
                         GPIO.output(Cool_Pin,True)
                         cold_status = 1
                         heat_status = 0
                    if (temp < desired_T - Ramp_Thresh): # Too
low below threshold, turn off A/C
                         GPIO.output(Heat_Pin,True)
                         GPIO.output(Cool_Pin,False)
                         cold_status = 0
                         heat_status = 1
                    print '{:.1f}'.format(time.time()-
initial_time),"Temp: ", '{:.1f}'.format(temp), "Desired Temp: ",
desired_T, "F, A/C Status ", cold_status, ", Heat Status ",
heat_status,  ", Cycles=", i,  ", Volts: ",
'{:.4f}'.format(volt)
                    fh = open("Log_Files/datalog%s.csv" %
gg,"a")
                    log_data = '{:.1f}'.format(time.time()-
start_time) + ',' + '{:.1f}'.format(temp) + ',' +
str(heat_status) + ',' + str(cold_status) + ',' +str(i) + ','  +
'{:.4f}'.format(volt) + ','  + '{:.4f}'.format(desired_T) + '\n'
                    fh.write(log_data)
                    fh.close()

#######--------#########-------###########---------
###########
          print 'Cool Down Complete, holding temperature for 3
hours'
          wait_time = time.time() # initializing timer for
waiting period
          while (time.time() < wait_time + 10800.0): #10800
seconds = 3 hours
               if (time.time() - prev_time) >= 60.0: # 60 sec
update rate
                    desired_T = Tmin # desired temp at current
time
                    [temp, volt] = temperature_reading()
                    if (temp > desired_T + Cold_Thresh): # Too
high above threshold, need to cool
                         GPIO.output(Heat_Pin,False)
```

```
                        GPIO.output(Cool_Pin,True)
                        cold_status = 1
                        heat_status = 0
                    if (temp < desired_T - Cold_Thresh): # Too
low below threshold, turn off A/C
                        GPIO.output(Heat_Pin,True)
                        GPIO.output(Cool_Pin,False)
                        cold_status = 0
                        heat_status = 1
                    print '{:.1f}'.format(time.time()-
initial_time),"Temp: ", '{:.1f}'.format(temp), "Desired Temp: ",
desired_T, "F, A/C Status ", cold_status, ", Heat Status ",
heat_status,  ", Cycles=", i, ", Volts: ", '{:.4f}'.format(volt)
                    fh = open("Log_Files/datalog%s.csv" %
gg,"a")
                    log_data = '{:.1f}'.format(time.time()-
start_time) + ',' + '{:.1f}'.format(temp) + ',' +
str(heat_status) + ',' + str(cold_status) + ',' +str(i) + ','  +
'{:.4f}'.format(volt) + ','  + '{:.4f}'.format(desired_T) + '\n'
                    fh.write(log_data)
                    fh.close()

#######---------#########-------###########----------
###########
          print 'Wait Complete, Heating up'
          ramp_time = time.time() # Another timer dummy variable
for relative timing
          wait_time = time.time() # timer for 4 hour ramp
          while (time.time() < wait_time + 14400): # 4 hour time
for ramp
               if (time.time() - prev_time) >= 60.0: # 60 sec
update rate
                    cycle_time = time.time() - ramp_time #this
gives time since start of ramp
                    desired_T = Tmin + ((Tmax -
Tmin)*cycle_time*0.25/3600.0) # desired temp at current time
                    [temp, volt] = temperature_reading()
                    if (temp > desired_T + Ramp_Thresh): # Too
high above threshold, turn off heat
                        GPIO.output(Heat_Pin,False)
                        GPIO.output(Cool_Pin,True)
                        cold_status = 1
                        heat_status = 0
                    if (temp < desired_T - Ramp_Thresh): # Too
low below threshold, turn on heat
                        GPIO.output(Heat_Pin,True)
                        GPIO.output(Cool_Pin,False)
```

```
                         cold_status = 0
                         heat_status = 1
                    print '{:.1f}'.format(time.time()-
initial_time),"Temp: ", '{:.1f}'.format(temp), "Desired Temp: ",
desired_T, "F, A/C Status ", cold_status, ", Heat Status ",
heat_status,  ", Cycles=", i, '{:.4f}'.format(volt)
                    fh = open("Log_Files/datalog%s.csv" %
gg,"a")
                    log_data = '{:.1f}'.format(time.time()-
start_time) + ',' + '{:.1f}'.format(temp) + ',' +
str(heat_status) + ',' + str(cold_status) + ',' +str(i) + ','  +
'{:.4f}'.format(volt) + ','  + '{:.4f}'.format(desired_T) + '\n'
                    fh.write(log_data)
                    fh.close()

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

          print 'Heat Up Complete, holding temperature for 1
hour'
          wait_time = time.time() # initializing timer for
waiting period
          while (time.time() < wait_time + 3600.0): #3600
seconds = 1 hour
               if (time.time() - prev_time) >= 60.0: # 60 sec
update rate
                    desired_T = Tmax # desired temp at current
time
                    [temp, volt] = temperature_reading()
                    if (temp > desired_T + Hot_Thresh): # Too
high above threshold, need to cool
                         GPIO.output(Heat_Pin,False)
                         GPIO.output(Cool_Pin,True)
                         cold_status = 1
                         heat_status = 0
                    if (temp < desired_T - Hot_Thresh): # Too
low below threshold, turn off A/C
                         GPIO.output(Heat_Pin,True)
                         GPIO.output(Cool_Pin,False)
                         cold_status = 0
                         heat_status = 1
                    print '{:.1f}'.format(time.time()-
initial_time),"Temp: ", '{:.1f}'.format(temp), "Desired Temp: ",
desired_T, "F, A/C Status ", cold_status, ", Heat Status ",
heat_status,  ", Cycles=", i, '{:.4f}'.format(volt)
                    fh = open("Log_Files/datalog%s.csv" %
gg,"a")
```

```
                    log_data = '{:.1f}'.format(time.time()-
start_time) + ',' + '{:.1f}'.format(temp) + ',' +
str(heat_status) + ',' + str(cold_status) + ',' +str(i) + ','  +
'{:.4f}'.format(volt) + ','  + '{:.4f}'.format(desired_T) + '\n'
                    fh.write(log_data)
                    fh.close()
#######---------#########-------###########----------
###########

     # End of ramp test, turn off all systems #
     GPIO.output(Heat_Pin,False)
     GPIO.output(Cool_Pin,False)
```