Analysis of the Ultimate Dew Shield Temperature Control Loop

C.Y. Tan*

(Dated: January 30, 2016)

Abstract

The design of the temperature control loop of the *Ultimate Dew Shield* is discussed here. The design started with P and PI loops simulations using the measured step response of the system. These results of these simulations determine the choice of loop and the gain of the loop. Finally, the measurement of the closed loop response is compared with the simulations to validate both the final implementation and simulations.

* cytan299@yahoo.com

I. INTRODUCTION

This is the document that discusses how I designed the temperature control loop for the *Ultimate Dew Shield*[1]. Most temperature controllers will use a PI (proportional, integral) loop. But I decided that, rather than just blindly use it, it is better for me to choose between P or PI loops. This means that I have to be able to simulate the P and PI loop of my system. And to do this, I have to measure the step response of the system and use this data in the simulation. Once I decide from the simulation which loop to use, I can test the temperature controller with the chosen loop in the real world. Finally, to literally close the loop of my understanding of the system, I can compare the temperature controller's real world performance with the simulation that decided its type.

Thus, the first step is for me is to measure the step response of the system, which I will discuss below.

II. STEP RESPONSES

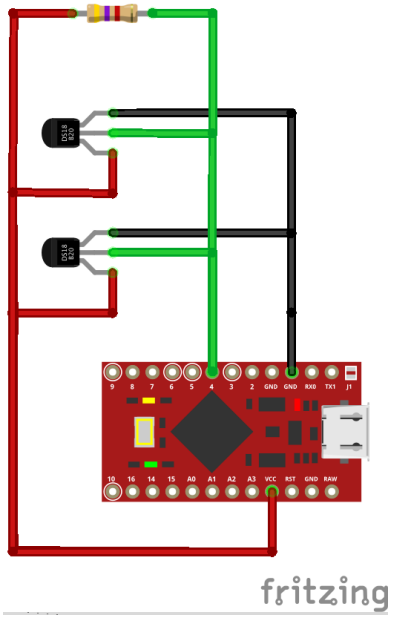

FIG. 1. This is the simple circuit used for measuring the step response. It consists of one Arduino Pro Micro, two DS18B20's and a 4.7kΩ resistor. Power comes from the USB connection.

For the step response, I had to create a temporary set up to measure it because I did not, at that time, have the electronics completely made yet. It consists of two temperature

probes, DS18B20,[2] called A and B, and an Arduino Pro Micro[3]. My circuit is shown in Fig. 1, and it is obviously very simple. My code that reads the temperature probes and sends the required data back to my Mac is shown in Appendix A.

Fig. 2 shows how I jerry-rigged up the probes and the Arduino to the dew shield for my measurement. I mounted the temperature probe A inside the dew shield as close as possible to the corrector plate of my LX200. This is the best position to measure the temperature of the air that insulates the corrector plate from the external cold air. For probe B, which measures the ambient temperature, I just let it hang outside the dew shield. Data is collected approximately once a second.

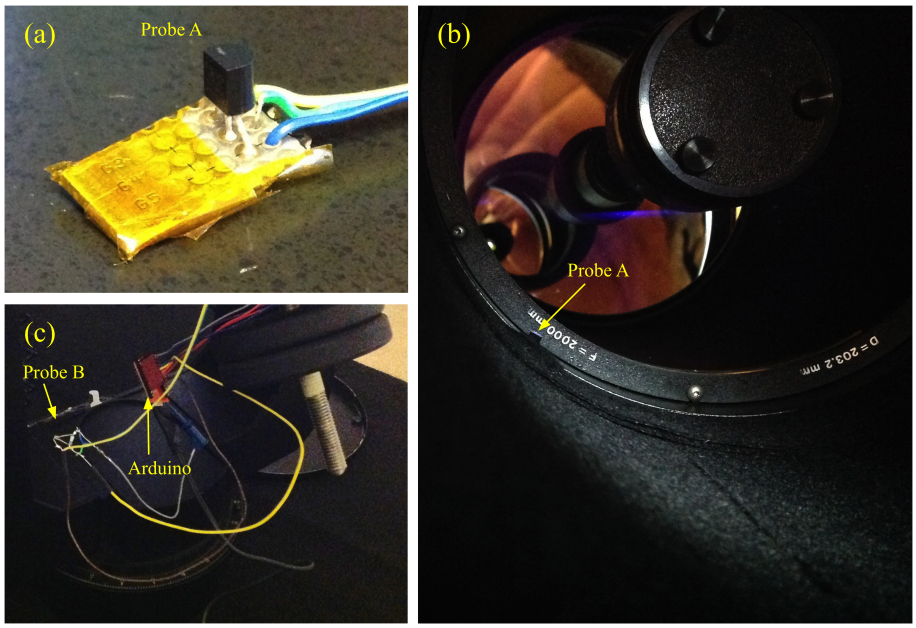

FIG. 2. This is how I jerry rigged up a test setup for measuring the step response. (a) is how I soldered the temperature probe to a perf board, and (b) shows where I inserted the it. (c) The ambient temperature probe is left hanging outside the dew shield.

The step response of the LX200 and dewshield measured from this setup is shown in Fig. 3. I measured both the heating response, i.e. when I turn on the dew shield heaters, and its cooling response, i.e. when I turn off the dew shield. I can fit the heating and cooling

responses with exponentials to get the time constants. The fits that I have found are

$$\mathcal{T}_{\text{heating}}(t) = 3.9833(1 - e^{(315.216-t)/446.635}) \tag{1}$$

$$\mathcal{T}_{\text{cooling}}(t) = 0.0563411 + 380.779e^{(1723.11-t)/210.207} \tag{2}$$

Thus, the heating time constant is 446 s and the cooling time constant is 210 s. These two time constants are quite long and affects the performance of the temperature control loop.

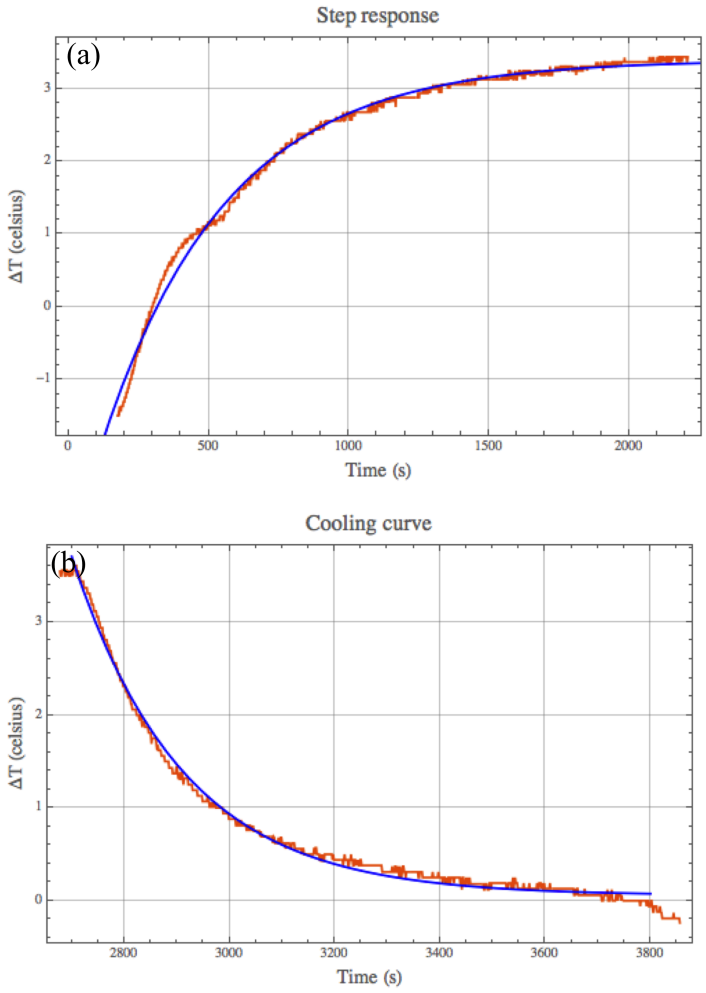

FIG. 3. These are the responses that I measured (a) is the heating response and (b) is the cooling response of the system. The measurements are in blue while the fits are in red.

III. SIMULATIONS

In order for me to decide between the P and PI loop, I have to simulate how the temperature controller behaves with either of these loops using the heating and cooling responses that I have measured in the previous section. The block diagram that I have used in the simulation is shown in Fig. 4.

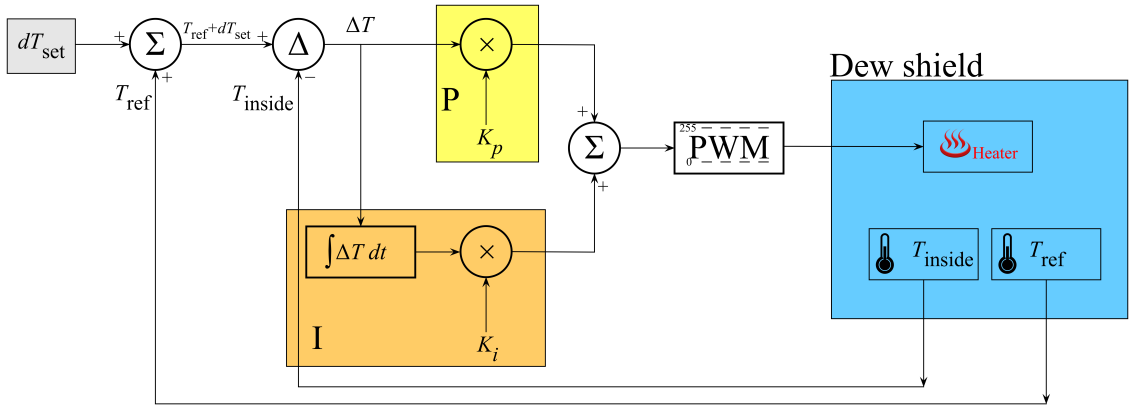

FIG. 4. The block diagram of the P and PI loops used in my simulation.

Using this block diagram, I can trace the loop . Starting from the left, I set the temperature offset dT_{set} from T_{ref} where I want the temperature dew shield to be, i.e.

$$T_{\text{set}} = T_{\text{ref}} + dT_{\text{set}} \tag{3}$$

T_{ref} in this case is either the ambient temperature T_{outside} or the dew point T_{dewpoint}. The difference between the measured temperature T_{inside} of the dew shield and T_{set} gives me the error of the control loop ΔT.

A. P loop

Now, if the gain K_i of the "I" part of the loop is zero, then I only have the "P" part of the loop. I set K_p to a non-zero value and the product $K_p\Delta T$ is sent to the pulse width modulator (PWM). The PWM sets the amount of dew shield heating and for the Arduino, the range of the PWM is between 0 and 255 in integer steps. Thus if $K_p\Delta T > 255$, then I set PWM $= 255$. And if $K_p\Delta T < 0$, then I set PWM $= 0$.

And after 1 s, the measured temperatures T_{inside} and T_{ref} have new values and these are fed back to the beginning of the loop.

B. PI loop

I have a PI loop if $K_i \neq 0$. In this case ΔT is integrated in time and multiplied by K_i. Since I sample the temperatures every 1 s, I can transform the integral into a sum, i.e.

$$K_i \int_0^\infty \Delta T \, dt \quad \rightarrow \quad K_i \sum_{j=0}^{\infty} \Delta T \times \Delta t \tag{4}$$

The above is summed with $K_p \Delta T$ from the P loop and the result is sent to the PWM.

C. Using the step responses

The exponential heating and cooling equations that I have used in the simulations are

$$T_{\text{heating}}(t) = T_h(1 - e^{-t/\tau_h}) \tag{5}$$

$$T_{\text{cooling}}(t) = T_c e^{-t/\tau_c} \tag{6}$$

where $T_{\text{heating}}(t)$ is the temperature of the dew shield at time t after the dew shield heater is turned on, T_h is the asymptotic temperature that the dew shield reaches as $t \rightarrow \infty$ and τ_h is the heating time constant; $T_{\text{cooling}}(t)$ is the temperature at time t of the dew shield after the heaters are turned off, T_c is the initial temperature of the dew shield instantaneously before the heaters are turned off, and τ_c is the cooling time constant.

I can linearize the above equations so that I can calculate the temperature rise of the dew shield after time Δt. When I do this, I have

$$\Delta T_{\text{heating}}(\Delta t) = \frac{T_h}{\tau_h} e^{-\Delta t/\tau_h} \Delta t \tag{7}$$

$$\Delta T_{\text{cooling}}(\Delta t) = -\frac{T_c}{\tau_c} e^{-\Delta t/\tau_c} \Delta t \tag{8}$$

From section II, I have $T_h = 5°\text{C}$, $\tau_h = 446$ s and $\tau_c = 210$ s. But for the heating equation, this is for full power, i.e. it is equivalent to PWM = 255. Thus, if I assume that $\Delta T_{\text{heating}}$ scales linearly as the value of the PWM, then I can modify Eq. 7 to get

$$\Delta T_{\text{heating}}(\Delta t, \text{PWM}) = \frac{\text{PWM}}{255} \frac{T_h}{\tau_h} e^{-\Delta t/\tau_h} \Delta t \tag{9}$$

D. Results

The relevant fragments of my simulation programs written in *Mathematica* for both the P and PI loops are in Appendix B and C. I performed the simulation for $T_{\text{set}} = 2$ and a few values of K_p and K_i for $K_p = 1024$ that are shown in Fig. 5. And to make things a little more realistic, I have also added white noise to ΔT to simulate noise at the 0.0625°C level because this is the least significant bit of the DS18B20.

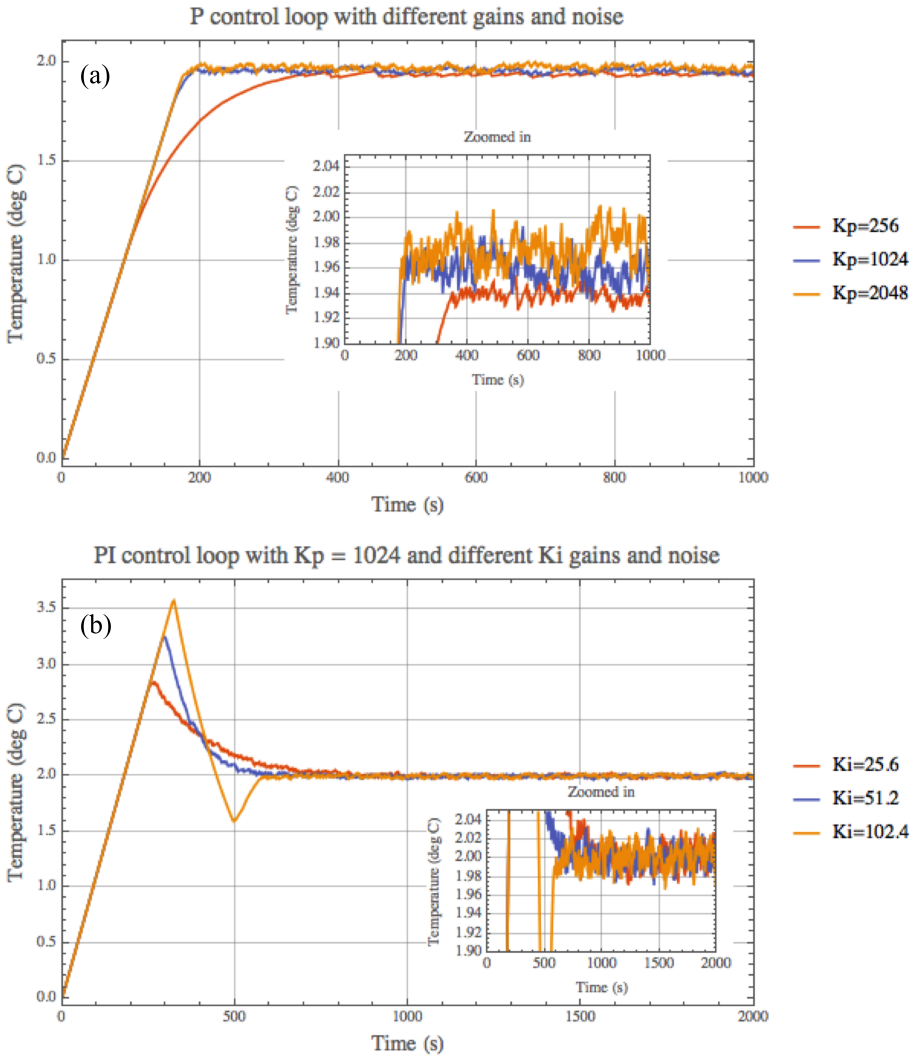

FIG. 5. The results of the simulations for the P and PI loops. (a) is shows the behaviour of the P loop and (b) is the behaviour of the PI loop.

It is obvious from Fig. 5(a) that the P loop never quite reach "2", but it gets closer to this value he higher the value of K_p is. This is, of course, what I'd expected from control

loop theory.[4] From the simulation, I can see that $K_p \sim 1024$ is about the right value for the P loop.

The usual fix for getting the control loop to eventually get to its target setting is is to add in the "I" loop. My simulation of $K_p = 1024$ and a few K_i values are shown in Fig. 5(b). This time, clearly, the PI loop eventually arrives at "2". However, the addition of the "I' loop is not free: this loop overshoots the target value before settling into "2". The overshoot gets worse as K_i gets larger.

After seeing these results, I decided that the overshoot with the PI loop is just not worth it because the settling time is actually quite long, of order 500 seconds. Besides, the accuracy of the DS18B20 is ± 0.5°C and thus as long as $|\Delta T| < 0.5$°C, I decided that it is good enough for the temperature regulation of the dew shield. Therefore, I have chosen to just use the P loop and have set the gain $K_p = 1024$ in the final production.

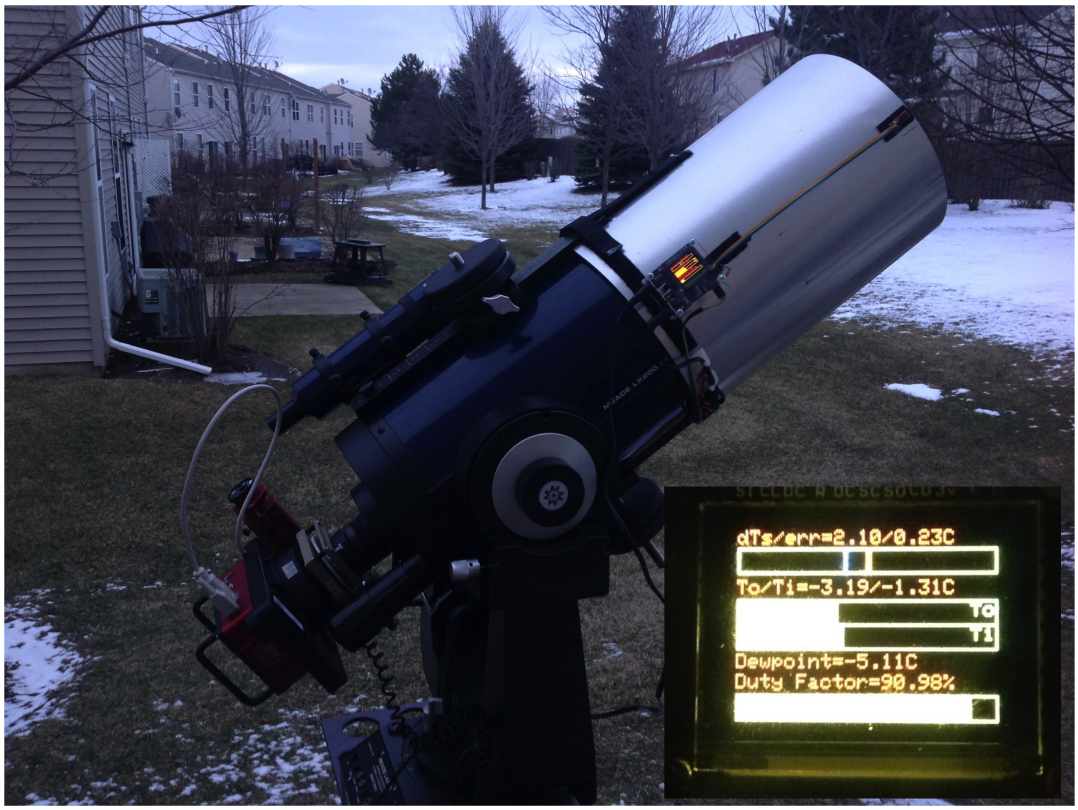

FIG. 6. The temperature controller installed on the dew shield, which in turn is installed on my LX200. The inset shows the display of the controller.

IV. PERFORMANCE

Once I finished building the controller, I wanted to see how it performs. The finished controller installed on the dew shield is shown in Fig. 6. The details of its construction can be found in [5].

I have built the controller so that serial data can be uploaded to my Mac. Fig. 7 shows the controller tracking $(T_{\text{outside}} + 2)$°C clearly with some error. However, the error is always < 0.5°C which is my design specification discussed in section III D.

A. Comparison with simulation

To close the "loop" between simulation results and real world measurements, I compare the measured and simulated T_{inside} shown in Fig. 7. The simulation program that I have used is a slightly modified one shown in Appendix B.

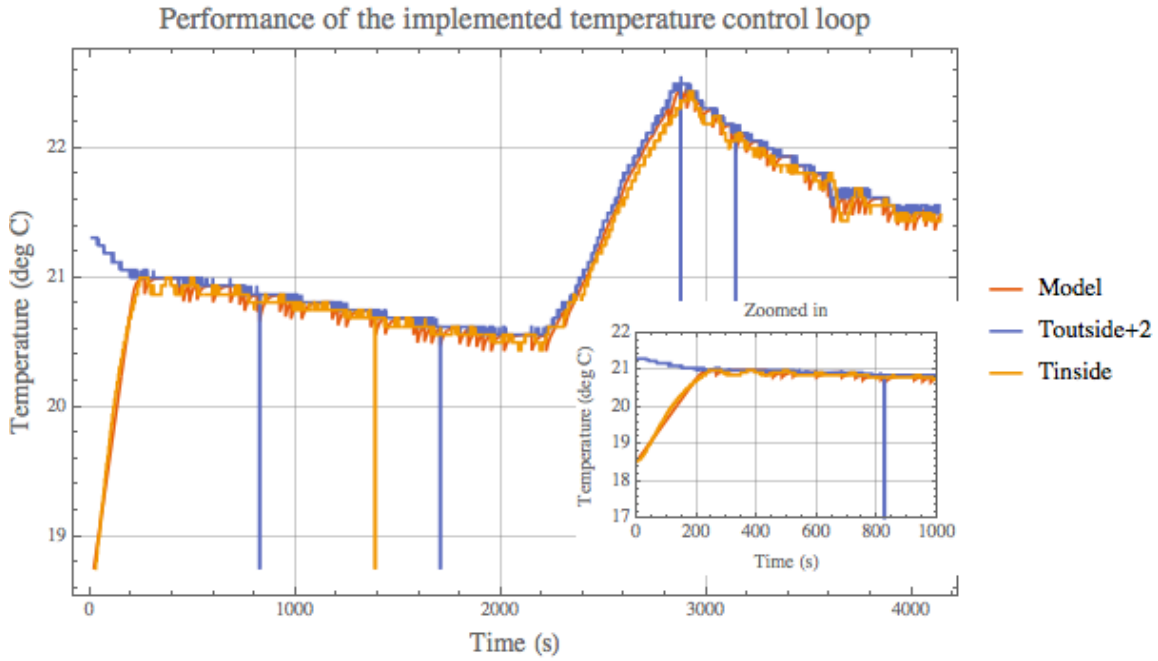

FIG. 7. The "real world" performance of the control loop compared to the simulated loop. The inset shows the zoomed in view of the loop heating up the dew shield to get it to the desired temperature.

Although the measured results, in the general, look similar to the simulation results, I decided to look closer. I can plot the loop errors ΔT in these two cases and this is shown

in Fig. 8. It clearly says that the simulated loop performs better than the measured one because the loop error is approximately 0.2°C worse, on average, than the model.

I can, of course, spend more time tracking down where the difference comes from, but I'm going to call it good for now.

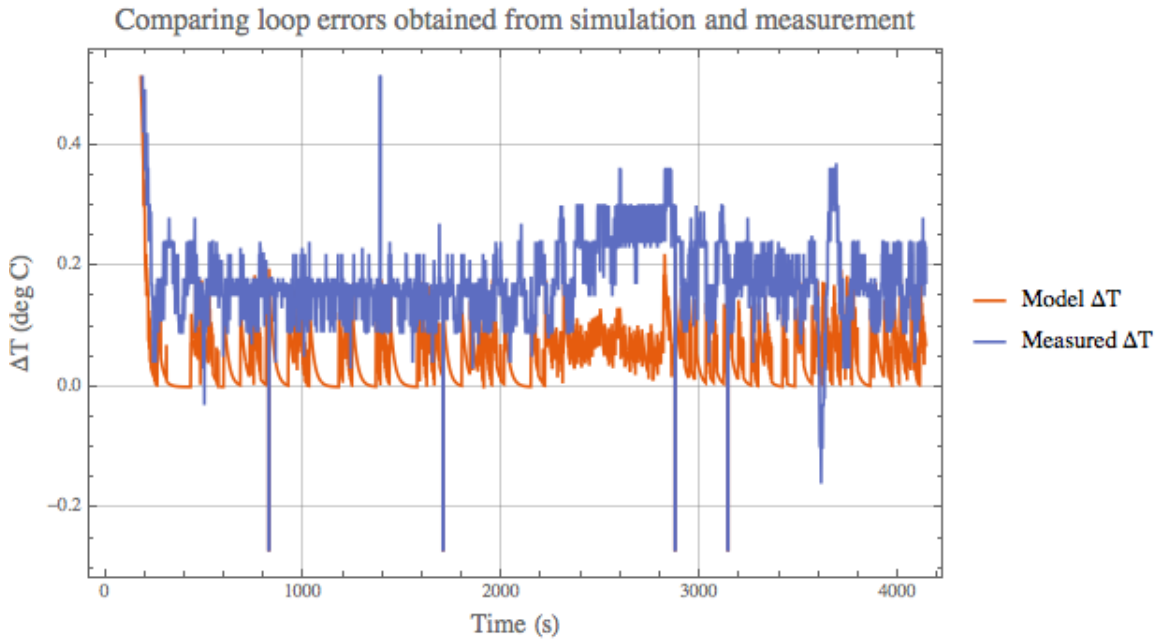

FIG. 8. The measured and simulated ΔT's are plotted here. It looks like the simulation says that the loop can perform better than what I have measured.

V. CONCLUSION

I have completed the analysis of the dew shield temperature controller. It is an interesting exercise for me to show how, from the step response of the system, I can

- decide on the type of loop
- decide on the gain of the loop
- show that, in general, the control loop behaves as expected.

Have fun analyzing your own temperature control loop when you build your *Ultimate Dew Shield*!

Appendix A: Code for measuring the step response

```
#include "OneWire.h"
#include "DallasTemperature.h"

/*********************************************************
  Defines for the thermometers
*********************************************************/
/* Pin defines for DS18B20 temperature probes */
#define temp_pin        4

OneWire onewire_temp(temp_pin);
DallasTemperature temp_probe(&onewire_temp);

void setup()
{

  Serial.begin(115200);
  while(!Serial);
  Serial.println(F("Serial port is open"));

  // DS18B20 code initialize
  temp_probe.begin();
  temp_probe.setWaitForConversion(false); // set to non-blocking
}

// define the time to request and to get the temperature probe data
#define TEMP_REQUEST_TIME       500 // ms
#define TEMP_GET_TIME          1000 // ms

void loop()
{
```

```
  static unsigned long time_ms = millis();
  static bool is_request_temperature = true;

  int dt = millis() - time_ms;
  if((dt < TEMP_REQUEST_TIME) && is_request_temperature){
    temp_probe.requestTemperatures();
    is_request_temperature = false;
  }

  if(dt > TEMP_GET_TIME){
    float Toutside = temp_probe.getTempCByIndex(0);
    float Tinside = temp_probe.getTempCByIndex(1);

    time_ms = millis();

    Serial.print(time_ms); Serial.print(" ");
    Serial.print(Tinside); Serial.print(" ");
    Serial.println(Toutside);

    is_request_temperature = true;
  }

}
```

Appendix B: P Loop Mathematica code

```
dTrise[pwm_, dt_] := pwm 5/446 Exp[-dt/446] dt
dTcool[T0_, dt_] := -T0/210 Exp[-dt/210] dt

DT[Tset_, Tin_, isNoise_]:=(Tset - Tin) +
                            If[isNoise, RandomReal[{-1, 1}] 0.06255, 0];

PLoop[Kp_, Tset_, Ti_, isNoise_] := Module[{L, Lpwm, dT, pwm, Tin, i},
   Tin = Ti;
   L = {{0, Tin}};
   Lpwm = {{0, 0}};
   For[i = 1, i <= 1000, i++,
    dT = DT[Tset, Tin, isNoise];
    pwm = Kp dT;
    (*pwm is granular and takes fractions between 0 and 255 in steps \
of 1/256*)
    pwm = IntegerPart[pwm 255]/256;
    (*Check that Kp dT < 1 or else set to max*)

    If[pwm > 1, pwm = 1];
    If[pwm > 0,
     Tin = Tin + dTrise[pwm, dt] // N,
     Tin = Tin + dTcool[Tin, dt] // N;
     pwm = 0;
     ];
    L = Join[L, {{i dt, Tin}}];
    Lpwm = Join[Lpwm, {{i dt, pwm}}];
    ];
   Return[{L, Lpwm}];
   ];
```

Appendix C: PI Loop Mathematica code

```
PILoop[Kp_, Ki_, Tset_, Ti_, isNoise_] :=
  Module[{L, Lpwm, dT, pwm, Tin, i, sumT},
   Tin = Ti;
   L = {{0, Tin}};
   Lpwm = {{0, 0}};
   sumT = 0;
   For[i = 1, i <= 2000, i++,
    dT = DT[Tset, Tin, isNoise];
    sumT += dT;
    pwm = Kp dT + Ki sumT;
    (*pwm is granular and takes fractions between 0 and 255 in steps \
of 1/256*)
    pwm = IntegerPart[pwm 255]/256;
    (*Check that Kp dT < 1 or else set to max*)

    If[pwm > 1, pwm = 1];
    If[pwm > 0,
     Tin = Tin + dTrise[pwm, dt] // N,
     Tin = Tin + dTcool[Tin, dt] // N;
     pwm = 0;
     ];
    L = Join[L, {{i dt, Tin}}];
    Lpwm = Join[Lpwm, {{i dt, pwm}}];
    ];
   Return[{L, Lpwm}];
   ];
```

[1] C.Y. Tan. The Ultimate Dew Shield. https://github.com/cytan299/ultimate_dew_shield, 2016. [Online].

[2] Maxim Integrated. DS18B20 Programmable Resolution 1-Wire Digital Thermometer. http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf, 2008. [Online].

[3] Sparkfun. Arduino Pro Micro. https://www.sparkfun.com/products/12640, 2014. [Online].

[4] Wikipedia. PID controller. https://en.wikipedia.org/wiki/PID_controller. [Online].

[5] C.Y. Tan. How to build the Ultimate Dew Shield. https://github.com/cytan299/ultimate_dew_shield/tree/master/howto, 2016. [Online].