Shower Temperature Controller

By Rees Shephard Parker, 25593417

Description of the product:

The shower in my apartment has erratic water temperatures. Without moving the knob, the water will rapidly become scalding hot or tap cold. Through a series of rudimentary tests while showering, I determined that this is due to some inconsistent pressures in the plumbing as there is always hot water when the knob is turned fully CW. The resulting randomly shifting water temperature in my apartment's shower leads to a poor showering experience, as such, I thought this would be a good opportunity to learn more about stepper motors, temperature sensors, and designing control systems. This shower temperature controller uses a thermistor in line with the plumbing to determine water temperature and a stepper motor to turn the shower knob to modify water temperature.

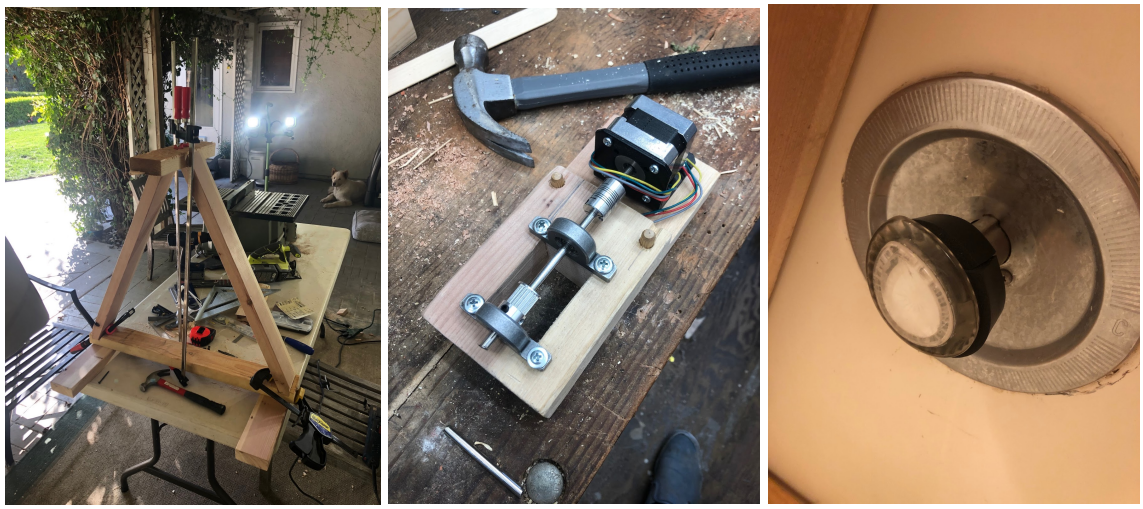
[Figure 1]: The shower temperature controller, note the high-tec seran-wrap and duct-tape-egg-carton drip guards to protect the breadboard and electronics.

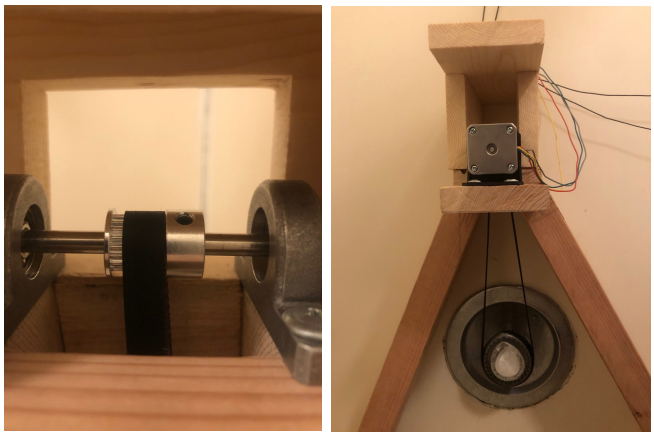
Too see a demo of the product running, see the provided link: https://youtu.be/gSVHAPVClmM

Electromechanical details: (~1 page total with images)

Mount/Frame/Housing: The motor and power transmission sits atop a wood frame. There is nothing fancy about this wood frame, I built it simply because I am a renter and prefer keeping my security deposit intact. A more permanent version of this controller would most likely be screw-mounted or embedded in the walls. The roof and walls (the lighter colored wood) above the motor is removable for easy access to power transmission.

Power Transmission: The main output power source is obviously the stepper motor (https://www.adafruit.com/product/324), but to get that power to the shower knob required a shaft (https://www.mcmaster.com/1265K44/), flexible shaft coupling (https://www.mcmaster.com/2464K1/), bearings (https://www.mcmaster.com/6153K113/), bearing mounts (https://www.mcmaster.com/3971N52/), a timing pulley (https://www.mcmaster.com/3684N13/), and a timing belt (https://www.mcmaster.com/3682N2/). Additionally, the shower knob was wrapped with sponge window sealing tape to increase grip between the knob and timing belt. The timing belt pulley had to be measured and put together using super glue and satin-ribbon.





[Figure 2]: Going left to right, top to bottom, we have the wood frame mid-assembly, the motor and power transmission (note all the individual parts), the shower knob wrapped in sponge sealant, the timing belt around the timing pulley in line with the motor, and the front view of the stepper-knob power transmission.

Thermistor: The thermistor (PANW103395-395) was put in line with the shower head so that it could non-invasively read water temperature during operation. Putting it in line was just a matter of using various adapters and teflon tape to prevent leaks. The thermistor was then put in series with a 10 kΩ resistor and analog output was placed between them. A higher water temperature led to a lower resistance which led to lower output recorded voltage. Cold led to higher which led to higher.

[Figure 3]: From left to right, the thermistor alone, installed in plumbing adapters, and, finally, put in line with the shower head.

Circuit and system response:

This project has three main components, the stepper motor, the thermistor, and the stepper response.

1. Stepper: To determine what stepper motor I would need, I did some manual testing using a chopstick, rubber bands, and a very light weight (a small nalgene bottle weighing 90g) to determine that the amount of torque required to turn the shower knob was less than 0.1 N*m. I found a 12V stepper on adafruit (link is above) that could supply such torque even when powered with a 5V 2A DC/AC adapter. This stepper motor was then driven with the motor driver provided in the labkit, the DRV8833 Dual Motor Driver Carrier.
2. Thermistor: choosing a potential temperature sensor was a much simpler decision. I just googled thermocouples and thermistors until I found one that was capable of installing in a ⅛" plumbing adapter sold on digikey (link is above). This thermistor has a room temperature resistance of 10kΩ, so I placed it in series with the microcontroller's 3.3V output, a 10kΩ, and ground, and placed the analog output to pin A2 between the thermistor and resistor. Calibration data is available in Appendix 1. This data is from a test where I simply wired the thermistor and manually adjusted the shower temperature until it was comfortable. From this I determined that the target voltage reading that indicated a comfortable water temperature was in the range of 980 to 1000 mV. I could have used this voltage to find the resistance of the thermistor and used Appendix 3 to determine what temperature this corresponded to, but decided that simply if it was comfortable, that's all I needed to know (I won't lie, time crunch did factor into this decision).
3. Stepper response: my shower knob has a range of movement of 120º which corresponds to 810º of stepper motion. I determined this manually by measuring the shower knob max and min angles and seeing how far the stepper turned during this. From here I knew that 1º of knob rotation corresponds to 6.75º of stepper rotation. I first opted for an arbitrary set of turns as documented in Appendix 2, simply that the further the temperature is from the target, the more the stepper should turn. I found that this created a positive feedback loop as the stepper turned faster then the water could change temperature and the thermistor could report the change. I measured how long the water took to change temperature (~4 seconds), so I added a 5 second delay (to be safe) to the stepper response, the device was then able to maintain temperature within the 980-1000 mV range. A delay here is appropriate as changes in water temperature are relatively slow.

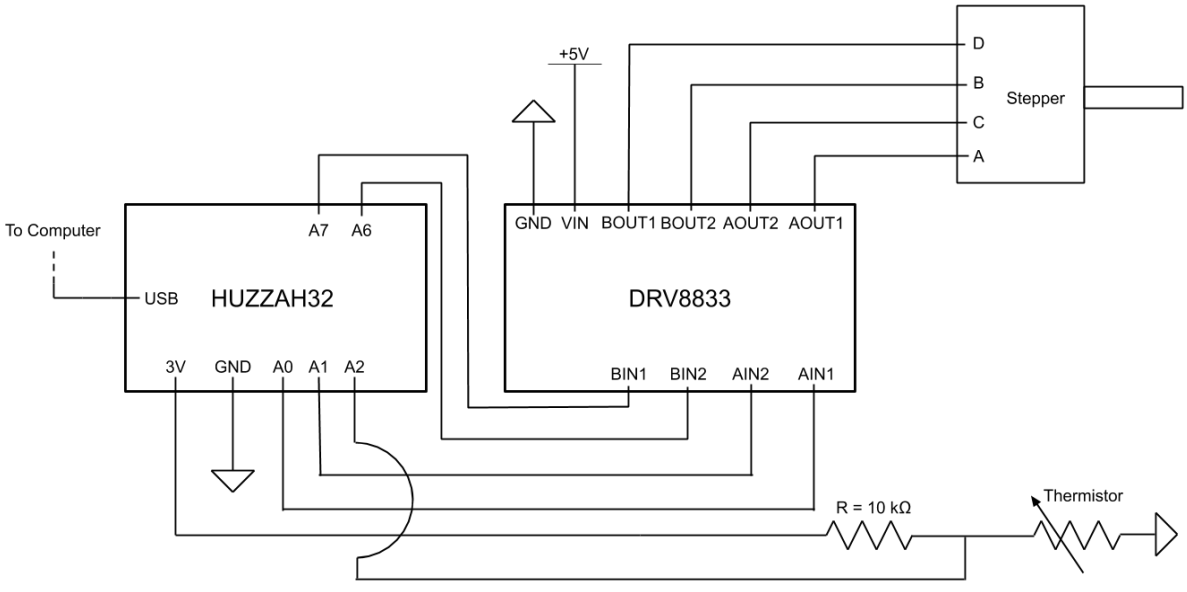


Figure 4: Wiring Diagram for the device.

Finite state machine:

The device's behavior is not too complicated. The micro simply checks temperature every 5 seconds and then drives the stepper if necessary. In Figure 5, it can be seen that the main complicating factor is just the specific response dictated by each voltage range. Also included is the graphed results from a test run.

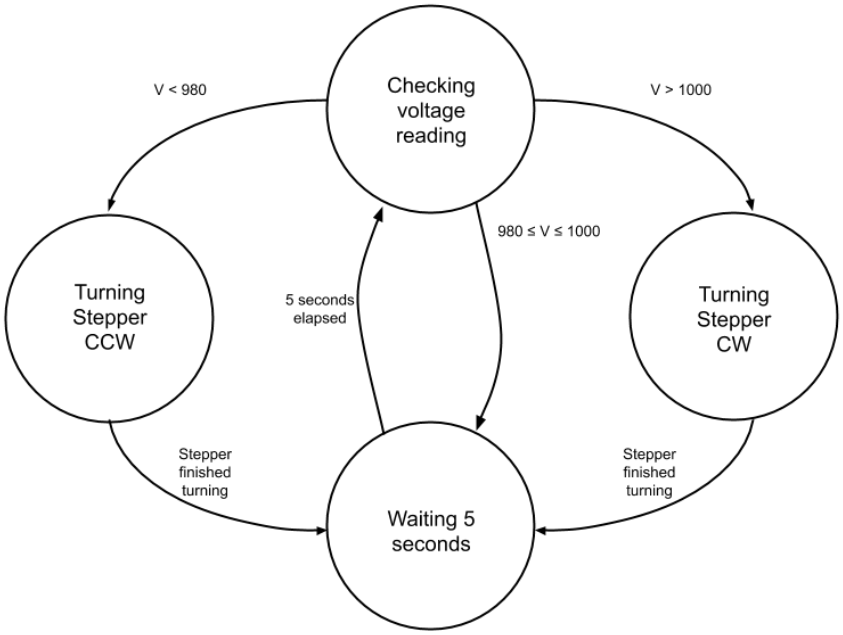


Figure 5: Finite State Diagram for the device.

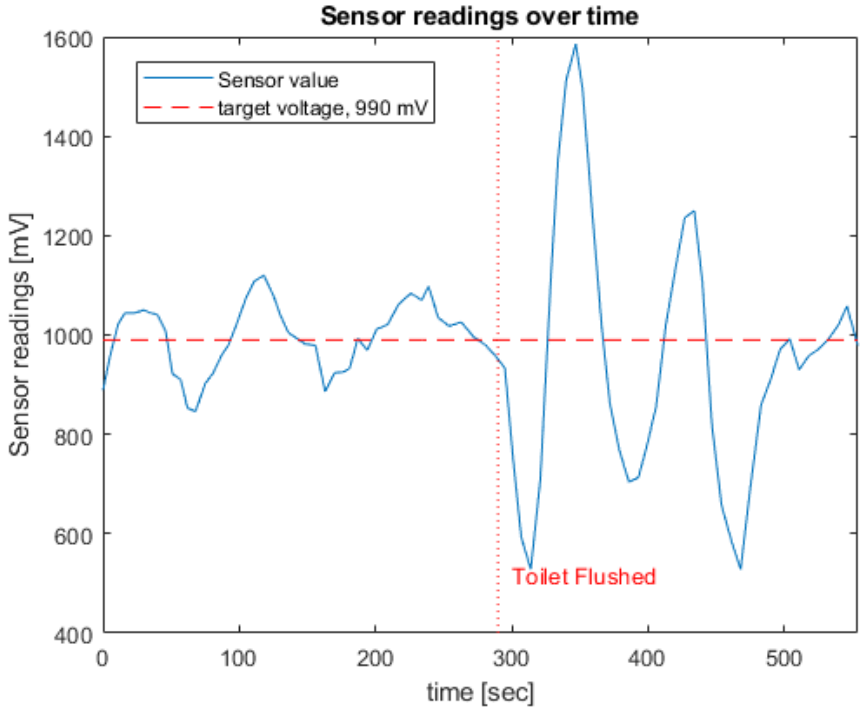


Figure 6: data from a run of the device. As can be seen above, the device is able to maintain a temperature around the corresponding ideal voltage of 980 mV. When a large disturbance is induced like a toilet flush, the device is able to return itself/the temperature to the ideal range around 980 mV.

Appendix 1: Calibration data with manual control to determine the comfortable ranges for the device to maintain. It took ~150 seconds for the shower to heat up and for me to initially find the ideal shower temp.

Timestamp		Sensor Value [mV]	Elapsed Time [s]
20:42:08	->	979	149.5
20:42:09	->	1016	150
20:42:09	->	1012	150.5
20:42:10	->	1018	151
20:42:10	->	1000	151.5
20:42:11	->	999	152
20:42:11	->	992	152.5
20:42:12	->	1005	153
20:42:12	->	1005	153.5
20:42:13	->	1005	154
20:42:13	->	1017	154.5
20:42:14	->	992	155
20:42:14	->	1016	155.5
20:42:15	->	1014	156
20:42:15	->	1000	156.5
20:42:16	->	1005	157
20:42:16	->	1012	157.5
20:42:17	->	1005	158
20:42:17	->	996	158.5
20:42:18	->	1005	159
20:42:18	->	1004	159.5
20:42:19	->	992	160
20:42:19	->	1005	160.5
20:42:20	->	997	161
20:42:20	->	1004	161.5
20:42:21	->	992	162
20:42:21	->	1007	162.5
20:42:22	->	976	163
20:42:22	->	990	163.5
20:42:23	->	1006	164
20:42:23	->	984	164.5

20:42:24	->	992	165
20:42:24	->	987	165.5
20:42:25	->	992	166
20:42:25	->	979	166.5
20:42:26	->	1018	167
20:42:26	->	1006	167.5
20:42:27	->	987	168
20:42:27	->	992	168.5
20:42:28	->	1009	169
20:42:28	->	1001	169.5
20:42:29	->	988	170
20:42:29	->	994	170.5
20:42:30	->	996	171
20:42:30	->	984	171.5
20:42:31	->	987	172
20:42:31	->	986	172.5
20:42:32	->	983	173
20:42:32	->	979	173.5
20:42:33	->	988	174
20:42:33	->	985	174.5
20:42:34	->	967	175
20:42:34	->	984	175.5
20:42:35	->	995	176
20:42:35	->	979	176.5
20:42:36	->	976	177
20:42:36	->	932	177.5
20:42:37	->	996	178
20:42:37	->	984	178.5
20:42:38	->	994	179
20:42:38	->	967	179.5
20:42:39	->	971	180
20:42:39	->	989	180.5
20:42:40	->	992	181
20:42:40	->	997	181.5
20:42:41	->	981	182

20:42:41	->	992	182.5
20:42:42	->	987	183
20:42:42	->	983	183.5
20:42:43	->	992	184
20:42:43	->	970	184.5
20:42:44	->	992	185
20:42:44	->	982	185.5
20:42:45	->	948	186
20:42:45	->	987	186.5
20:42:46	->	974	187
20:42:46	->	979	187.5
20:42:47	->	979	188
20:42:47	->	982	188.5
20:42:48	->	979	189
20:42:48	->	979	189.5
20:42:49	->	979	190
20:42:49	->	979	190.5
20:42:50	->	979	191
20:42:50	->	978	191.5
20:42:51	->	994	192
20:42:51	->	983	192.5
20:42:52	->	983	193
20:42:52	->	987	193.5
20:42:53	->	981	194
20:42:53	->	994	194.5
20:42:54	->	987	195
20:42:54	->	987	195.5
20:42:55	->	980	196
20:42:55	->	987	196.5
20:42:56	->	990	197
20:42:56	->	994	197.5
20:42:57	->	975	198
20:42:57	->	974	198.5
20:42:58	->	971	199
20:42:58	->	971	199.5

20:42:59	->	983	200
20:42:59	->	971	200.5
20:43:00	->	987	201
20:43:00	->	988	201.5
20:43:01	->	992	202
20:43:01	->	975	202.5
20:43:02	->	995	203
20:43:02	->	994	203.5
20:43:03	->	979	204
20:43:03	->	971	204.5
20:43:04	->	984	205
20:43:04	->	971	205.5
20:43:05	->	971	206
20:43:05	->	993	206.5
20:43:06	->	971	207
20:43:06	->	1004	207.5
20:43:07	->	980	208
20:43:07	->	985	208.5
20:43:08	->	990	209
20:43:08	->	981	209.5
Mean		988.5123967	
Std		13.73870184	
Min	warm	932	
Max	tepid	1018	
All units in millivolts			

Appendix 2: A summation of a few tests and a breakdown of what the ideal stepper response shall be.

Test2	Max and Min Shower Temps			Degrees	Theoretical degrees	
	Max	1945		-60	-30	
	Min	573		60	30	

	Difference	1372		11.43333333	22.86666667	Volts/degree
Test3	Calibrating for Ideal Temps					
	Max	1018		Knob: Positive indicates CW		hotter
	Min	932		Knob: Negative indicates CCW		colder
	Mean	988.5123967				
	Std	13.73870184		Stepper: Positive indicates CCW		colder
				Stepper: Negative indicates CW		hotter
Conclusion						
Desired Range	Max	1000				
	Min	980				
	Mean	990				
Reaction						
	Signal [mV]	Knob [degrees]	Stepper [degrees]	rounded	---------->	really rounded
HOT	V < 600	-10	67.5	68	7	70
	600 <= V < 700	-8	54	54	5	50
	700 <= V < 800	-6	40.5	41	4	40
	800 <= V < 900	-4	27	27	3	30
	900 <= V < 980	-2	13.5	14	1	10
	980 <= V <= 1000	0	0	0	0	0
	1000 < V <= 1100	2	-13.5	-14	-1	-10
	1100 < V <= 1200	4	-27	-27	-3	-30
	1200 < V <= 1300	6	-40.5	-41	-4	-40
	1300 < V <= 1400	8	-54	-54	-5	-50
COLD	1400 < V <=	10	-67.5	-68	-7	-70

	1650					
COLDER	1650 < V <= 1900	20	-135	-135	-14	-140
COLDEST	1900 < V	30	-202.5	-203	-20	-200

Appendix 3: Thermistor Temperature Resistance Chart, R_{25} = 10kΩ. This is from the thermistor data sheet.

RESISTANCE VS. TEMPERATURE CURVE

Deg C	R_T/R_{25}	Deg C	R_T/R_{25}	Deg C	R_T/R_{25}	Deg C	R_T/R_{25}	Deg C	R_T/R_{25}	Deg C	R_T/R_{25}
-50	66.9745	-15	7.3476	20	1.2515	55	0.2948	90	0.0924	125	0.0353
-49	62.3986	-14	6.9470	21	1.1960	56	0.2844	91	0.0897	126	0.0344
-48	58.1649	-13	6.5704	22	1.1432	57	0.2743	92	0.0870	127	0.0336
-47	54.2458	-12	6.2164	23	1.0931	58	0.2646	93	0.0845	128	0.0328
-46	50.6159	-11	5.8834	24	1.0454	59	0.2554	94	0.0820	129	0.0320
-45	47.2520	-10	5.5700	25	1.0000	60	0.2465	95	0.0796	130	0.0312
-44	44.1331	-9	5.2751	26	0.9568	61	0.2379	96	0.0773	131	0.0304
-43	41.2398	-8	4.9975	27	0.9157	62	0.2297	97	0.0751	132	0.0297
-42	38.5544	-7	4.7359	28	0.8766	63	0.2219	98	0.0729	133	0.0290
-41	36.0608	-6	4.4895	29	0.8393	64	0.2143	99	0.0708	134	0.0283
-40	33.7440	-5	4.2572	30	0.8038	65	0.2070	100	0.0688	135	0.0277
-39	31.5905	-4	4.0382	31	0.7700	66	0.2001	101	0.0669	136	0.0270
-38	29.5877	-3	3.8317	32	0.7378	67	0.1933	102	0.0650	137	0.0264
-37	27.7243	-2	3.6368	33	0.7071	68	0.1869	103	0.0632	138	0.0258
-36	25.9897	-1	3.4529	34	0.6778	69	0.1807	104	0.0615	139	0.0252
-35	24.3743	0	3.2791	35	0.6498	70	0.1747	105	0.0598	140	0.0246
-34	22.8691	1	3.1165	36	0.6232	71	0.1690	106	0.0581	141	0.0240
-33	21.4660	2	2.9628	37	0.5978	72	0.1634	107	0.0566	142	0.0235
-32	20.1574	3	2.8176	38	0.5735	73	0.1581	108	0.0550	143	0.0230
-31	18.9365	4	2.6802	39	0.5503	74	0.1530	109	0.0535	144	0.0224
-30	17.7969	5	2.5504	40	0.5282	75	0.1481	110	0.0521	145	0.0219
-29	16.7327	6	2.4275	41	0.5071	76	0.1433	111	0.0507	146	0.0215
-28	15.7384	7	2.3111	42	0.4869	77	0.1388	112	0.0494	147	0.0210
-27	14.8091	8	2.2010	43	0.4677	78	0.1344	113	0.0481	148	0.0205
-26	13.9402	9	2.0968	44	0.4492	79	0.1301	114	0.0468	149	0.0201
-25	13.1273	10	1.9980	45	0.4316	80	0.1261	115	0.0456	150	0.0196
-24	12.3666	11	1.9044	46	0.4148	81	0.1221	116	0.0444		
-23	11.6544	12	1.8157	47	0.3987	82	0.1183	117	0.0433		
-22	10.9874	13	1.7315	48	0.3833	83	0.1147	118	0.0422		
-21	10.3624	14	1.6518	49	0.3686	84	0.1111	119	0.0411		
-20	9.7765	15	1.5761	50	0.3545	85	0.1077	120	0.0400		
-19	9.2271	16	1.5043	51	0.3415	86	0.1045	121	0.0390		
-18	8.7118	17	1.4361	52	0.3291	87	0.1013	122	0.0381		
-17	8.2281	18	1.3714	53	0.3172	88	0.0982	123	0.0371		
-16	7.7741	19	1.3099	54	0.3058	89	0.0953	124	0.0362		

DRAWN BY: C. Terry		AMETHERM Circuit Protection Thermistors
DATE: 4/19/16	REV: 3	
ORIG. M.Samii	APPR: M. Samii	NTC THERMISTOR PROBE
SHEET 2 of 2		PANW 103395-395

Appendix 4: The sensor readings for the run in the video, all units are in millivolts. Note that the flush occurs around 13:19:00.

Timestamp		Sensor Value [mV]
13:14:06	->	889
13:14:11	->	967
13:14:17	->	1021
13:14:22	->	1044
13:14:29	->	1044
13:14:36	->	1050
13:14:41	->	1044
13:14:46	->	1041
13:14:52	->	1008
13:14:57	->	922
13:15:03	->	910
13:15:08	->	853
13:15:14	->	846
13:15:21	->	902
13:15:27	->	924
13:15:33	->	959
13:15:38	->	979
13:15:44	->	1021
13:15:51	->	1075
13:15:57	->	1108
13:16:04	->	1120
13:16:11	->	1080
13:16:16	->	1041
13:16:22	->	1005
13:16:34	->	982
13:16:42	->	979
13:16:49	->	886
13:16:56	->	924
13:17:02	->	925
13:17:07	->	932
13:17:13	->	993
13:17:20	->	969

13:17:27	->	1012
13:17:35	->	1021
13:17:43	->	1062
13:17:52	->	1084
13:18:00	->	1070
13:18:05	->	1098
13:18:12	->	1035
13:18:20	->	1018
13:18:29	->	1026
13:18:38	->	996
13:18:47	->	979
13:18:54	->	958
13:19:01	->	933
13:19:07	->	751
13:19:13	->	593
13:19:20	->	528
13:19:27	->	709
13:19:35	->	1127
13:19:40	->	1353
13:19:46	->	1513
13:19:53	->	1586
13:19:58	->	1495
13:20:05	->	1251
13:20:12	->	1018
13:20:18	->	863
13:20:25	->	768
13:20:32	->	704
13:20:39	->	713
13:20:45	->	775
13:20:52	->	856
13:20:59	->	1018
13:21:06	->	1131
13:21:13	->	1236
13:21:20	->	1250
13:21:26	->	1105

13:21:33	->	817
13:21:40	->	657
13:21:47	->	588
13:21:54	->	528
13:22:01	->	690
13:22:09	->	859
13:22:16	->	910
13:22:23	->	971
13:22:30	->	992
13:22:37	->	930
13:22:44	->	958
13:22:51	->	971
13:22:58	->	990
13:23:06	->	1021
13:23:12	->	1058
13:23:20	->	979

Appendix 5: The Arduino Code. I do want to apologize that this is not in the "switch (state)" format, I was not aware what exactly that meant until I saw Professor Stuart's code in her example report, and by then I had already written this.

```
/*
  Senior Project: Shower Temperature Control
  - Shower Temperature Control uses a thermistor (analog input A4) to check
temperature and then a stepper motor
    (digital outputs A0,A1,A6,A7) to adjust the shower temperature knob to maintain
an ideal temperature.
  - As of now, the stepper inputs and ideal temps are done by feel according to the
"Thermistor Data" Google Sheet
    "totals" tab.

  Thermistor
  - PANW103395-395
  -
https://www.digikey.com/en/products/detail/PANW103395-395/570-1458-ND/9084083?itemSe
q=345736221
  - 10 kOhms at 25C.
  - voltage readings for my particular shower can be found at the above google
sheet.
  - voltage to temperature relationship is not linear, so temp analysis will be done
via Gooogle Sheets and MATLAB.
  - Note: lower resistance values means hotter, higher values means colder. Same
goes for sensor voltage values.
```

```
  Stepper
  - NEMA-17 size - 200 steps/rev, 12V 350mA
  - https://www.adafruit.com/product/324
  - 200 steps per revolution.
  - Through trial and error, I determnined the stepper motor can safely rotate at 10
RPM. Safely in this context
    simply meaning there is sufficient torque and no slippage.
  - Due to the wiring, myStepper.step(X > 0) is CCW, myStepper.step(X < 0) is CW.
  - Code is largely from
    -
https://www.tutorialspoint.com/arduino/arduino_stepper_motor.htm#:~:text=Advertiseme
nts,rotates%20in%20discrete%20step%20angles.

  Motor Driver
  - DRV8833 Dual Motor Driver Carrier
  - https://www.pololu.com/product/2130
  - Note: the above driver has a max input voltage of 11.8V so I'm using a 5V 2A
source.

  Stepper-Shower-Knob-Relationship
  - This function has to be started when the knob is vertical.
  - The shower knob has ~120 deg of play (+-60 deg from vertical).
  - The stepper motor has ~810 deg of play (+-405 from vertical).
  - To be safe, I will let the stepper rotate +-300 deg from vertical, approximately
+-44 deg for the knob.
  - CCW stepper leads to COLDER
  - CW  stepper leads to HOTTER

  NOTE: this sketch does not use a timer interrupt or delay to track the thermistor
readings. This is because the
        stepper function acts as a delay (the script stops running until the stepper
finishes moving), and this
        function only needs to track one input. For a more complex system, timer and
input interrupts should be
        used.

  ~ Rees Shephard Parker, Dec-07-2020 2:00am
*/

// the following is set-up for the stepper motor
#include <Stepper.h>

const int stepsPerRevolution = 200;
int currentStepperPosition = 0; //here, 0 degrees indicates vertical.
int stepperResponse = 0; //this will be determined later in the void loop.
// initialize the stepper library on pins A0,A1,A6,A7 for coil pins A,C,B,D
respectively.:
Stepper myStepper(stepsPerRevolution, A0,A1,A6,A7);

// the following is set-up for the thermistor.
// Note: thresholds are not defined here as there are multiple so that stepper
response does not create a
// positive feedback loop. Hysteresis thresholds are not necessary here.
```

```
const int sensorPin = A4; //our chosen analog input pin.
int sensorValue = 0; //will quickly be replaced.
int voltage = 990; //assume starting at nominal, will quickly be replaced.


void setup()
{
  // put your setup code here, to run once:

  // initialize the serial port:
  Serial.begin(9600);

  // stepper initialization:
  // set the speed at 5-10 RPMs:
  // Note: sometimes the stepper slips, sometimes it does not, regardless of RPMs,
but higher RPMs
  // do lead to more slippage.
  myStepper.setSpeed(5);
}

void loop()
{
  // put your main code here, to run repeatedly:

  // thermistor readings
  // read the value from the sensor: 0 is LOW, 4095 is HIGH
  sensorValue = analogRead(sensorPin);
  int voltage = map(sensorValue, 0, 4095, 0, 3300);

  // print commands are at the end of the loop.

  // the following "if" block is used to determine stepper response. See google
sheet for more details.
  if (voltage < 600)
  {
    stepperResponse = 70;
  }
  else if (voltage >= 600 && voltage < 700)
  {
    stepperResponse = 50;
  }
  else if (voltage >= 700 && voltage < 800)
  {
    stepperResponse = 40;
  }
  else if (voltage >= 800 && voltage < 900)
  {
    stepperResponse = 30;
  }
  else if (voltage >= 900 && voltage < 980)
  {
    stepperResponse = 10;
  }
```

```
  else if (voltage >= 980 && voltage <= 1000)
  {
    stepperResponse = 0;
  }
  else if (voltage > 1000 && voltage <= 1100)
  {
    stepperResponse = -10;
  }
  else if (voltage > 1100 && voltage <= 1200)
  {
    stepperResponse = -30;
  }
  else if (voltage > 1200 && voltage <= 1300)
  {
    stepperResponse = -40;
  }
  else if (voltage > 1300 && voltage <= 1400)
  {
    stepperResponse = -50;
  }
  else if (voltage > 1400 && voltage <= 1650)
  {
    stepperResponse = -70;
  }
  else if (voltage > 1650 && voltage <= 1900)
  {
    stepperResponse = -140;
  }
  else if (voltage > 1900)
  {
    stepperResponse = -200;
  }

  if (stepperResponse == 0)
  {
    // if temperature is within ideal range, wait half a second to not flood the
serial monitor.
    delay(5000);
  }
  else
  {
    // here, stepper needs to move and can move to modify temperature.

    // update position
    currentStepperPosition = currentStepperPosition + stepperResponse;

    // drive stepper, this functions like a delay().
    myStepper.step(stepperResponse);

    // by trial and error, my shower takes ~4 seconds to change temperature, so a 5
second delay
    // should ensure the next temperature reading is accurate to knob position.
    delay(5000);
```

```
  }

  // I had initially included a max stepper turn so that the stepper did not try to
overturn the knob,
  // but the stepper would sometimes slip or fail to turn which threw off the
position this script
  // thought it was at. I have left the max turn code below so that in the future it
could be re-
  // implemented if I got a stronger power source for the stepper.
  /*
  // track current stepper position.
  if (stepperResponse == 0)
  {
    // if temperature is within ideal range, wait half a second to not flood the
serial monitor.
    delay(5000);
  }
  else if ((currentStepperPosition + stepperResponse) >= -300 &&
(currentStepperPosition + stepperResponse) <= 300)
  {
    // here, stepper needs to move and can move to modify temperature.

    // update position
    currentStepperPosition = currentStepperPosition + stepperResponse;

    // drive stepper, this functions like a delay().
    myStepper.step(stepperResponse);

    // by trial and error, my shower takes ~4 seconds to change temperature, so a 5
second delay
    // should ensure the next temperature reading is accurate to knob position.
    delay(5000);
  }
  else
  {
    // if the knob/stepper have reached a max position and temeprature is still
outside nominal, just wait. Most
    // likely the system will have to be reset with the knob positioned better. This
situation should not occur.
    Serial.println("ERROR: stepper cannot rotate further.");
    delay(5000);
  }
  */

  // print the voltage reading so we can graph and determine temp later using above
google sheet.
  Serial.println(voltage);

  // the following print commands are for diagnostic purposes
  //Serial.print(voltage);
  //Serial.print(',');
  //Serial.print(stepperResponse);
  //Serial.print(',');
```

```
  //Serial.println(currentStepperPosition);
}
```