Stepper Motor and Temperature Controller with Arduinos

Thomas Li

Department of physics, Miami University

11/08/2018

Stepper Motor with Arduino

Introduction

A stepper motor is an electric device whose rotor performs stepwise rotations. Computer-driven stepper motors fall into the category of a motion-control positioning system; they are widely used in holding-positioning applications. In experimental physics, especially in the field of optics, they are used in precision positioning devices as with linear actuators and mirror mounts. In commercial uses, they are also used in various advanced optical areas such as image scanner and disc drives. [1]

One can drive the stepper motor directly by building a mechanical circuit with connection to it, but it would be much more convenient to take advantage of a microcontroller such as the Arduino whose functions could be designed by online coding and a controller chip that directly drives the stepper motor. The controller chip used in this experiment is *Allegro A4988*, the type of which has both *STEP* (step) and *DIR* (direction) pins to receive corresponding order signals made by a microcontroller, and four pin connections to a bipolar stepper motor, namely 1A, 1B, 2A, and 2B, where the number stands for each winding and the combination of two letters for a closed circuit loop. With the connection to a potentiometer (pot) powered with a 5V voltage supply from the breadboard, the overall circuit composed of these four major components is shown in Figure 1.

Step Size of the Motor

The very first step to achieve any type of control by an Arduino is to set up the arguments for the corresponding pin connections. In this experiment, pins A0 and 6 were set as the analog input for the pot and an externally connected button, while pins 2 and 13 were for the *DIR* and *STEP* pins on the controller chip.

The electric pulse used in this experiment was a rectangle one with 50% duty cycle, which means half of the period is active and the other half being inactive. To set up such pulse we used the *digitalWrite* function to set the step pin in high with a delay of x ms, followed by a low with the same delay time. Each *step(x)* order from Arduino generates such a rectangle pulse giving rise to a single step moving of the motor. The period for each step is therefore 2x ms.

We then used a loop function to count the total steps needed for a complete revolution that should be initiated by pressing the button. The signal of initiation is received when the button pin value read by *digitalRead* function becomes 1, meaning the button is pressed otherwise it should be 0. In order to count the steps automatically, after each stepping triggered by the pulse we set *count* = *count* +1 then re-read the button value. When one cycle is finished the *Serial.println(count)* will tell the total number of steps. The detailed steps with annotation of the coding is shown in Appendix 2 - Sketch 1.

By uploading the code to the Arduino and therefore to control the motor, we ran into some issues in the beginning. The motor was stuck before it finished one revolution that it was supposed to. It turned out that the motor device attached with a potentiometer only allowed certain amount of angular rotation, and we fixed it by turning it backwards and re-tested the steps.

We recorded ten data points for the number of steps per revolution, four of which being 200 steps, with others being 202, 205, 199, 198, 205, and 203, respectively. The number generally oscillates around the mean value 201.2, with the uncertainty being 2.441.

Backward-Forward Sketch

For the majority of applications in which a stepper motor could be employed, it is as important as making stepped rotations to rotate on both directions. One can make orders on the change in direction of the motor rotation by taking advantage of the *DIR* pin on the controller chip.

The initial programming setup with the pins, as well as the pulse pattern, is the same with the Step Size part of the experiment. In order to make oscillations on the motor pot voltage between 3V and 4V, since the angular position varying with the motor rotation determines the effective resistance of the pot and thus the voltage, the repeated turning of the rotation direction on the edge of the voltage range could achieve this. Since the analog input value for each volt is 200, which gives 3V as 600 and 4V as 800 correspondingly, we applied a loop function that make turns whenever the *analogRead* encounters these two values.

To achieve so we utilized two *if* functions in the loop. We named a parameter *potValue* to equal the voltage value converted from *analogRead* for clarity. After each *step(x)* order, the computer enters a *if* function and check the *potValue* each time. Once the value is less than 600 or greater than 800, it will first have a 3-second delay for a “break”, after which by using the *digitalWrite* on the corresponding pin Arduino sends a voltage signal to the *DIR* of the controller to change the direction of the motor. The detailed steps with annotation of the coding is shown in Appendix 2 - Sketch 2. As the code was well constructed, by uploading onto the Arduino and to the whole circuit the motor performed neat consistency with the expectation as it turned to an opposite direction whenever it reached 3V and 4V.

Summary

By taking advantage of the versatile microcontroller Arduino and the stepping feature of the stepper motor, we achieved stepper counting experiment that one can read the total counts of the motor simply on the program screen given a certain amount of time. This type of circuit could be used as a simple counter, whose period of each counting is completely dependent on the user. We also achieved the voltage oscillation experiment that kept the potentiometer voltage oscillating within a certain range. The capability of varying the voltage by the circuit itself is exceedingly important in many filed. For example, if an artist wants to decorate his room by dimming and brightening different colors of lights back and forth, the rate of change could be determined by the x value of the step, while the voltage oscillation could be achieved by this backward-forward sketch and circuit.

Temperature Controller with Arduino

Introduction

Temperature control plays a significant role in our daily life. From preserving a breakfast sandwich in the refrigerator to warming a whole building in a cold weather, we cannot live without the capability to control temperature, particularly maintaining the temperature in a certain range. Due to versatility, electric circuits have an absolute advantage in achieving the goal. Generally, the practical approach to maintain a temperature is not by creating a heat bath to provide thermal conduction with the environment, but by repeatedly turning on and off the heating/cooling process whenever the environment reaches a limit temperature.

In circuitry, the most basic elements to perform such a purpose consist in three major parts: a temperature sensor, a circuit controller, and a heating/cooling device. Since for most

purposes temperature maintaining often only allows a small variation, a good sensitivity is required for the temperature sensor. Due to the internal structure, a semiconducting material often bears a good sensitivity as its resistance varies significantly with the temperature. Sensors made of this type of material are called thermistors. It is theoretically known that the relationship between resistance of the thermistor $R_{thermistor}$ and the absolute temperature T follows $R_{thermistor}= Ae^{B/T}$, where A and B are constants determined by the thermistor.

Experimental Setup - Hardware

Before starting the temperature control experiment, it is important to know the characteristic constants of the thermistor and the relation to the circuit. The circuit we used for this experiment is shown in Figure 2. The thermistor we used was essentially two wires wrapped by a rubber protection, one side being the pins to be connected to the circuit and the other as the sensor to be contacted with the environment of interest. The op-amp we used is of 747 type that is composed of two internal op-amps, the first of which is connected such that it forms a inverting op-amp serving as an ohmmeter, while the second one is connected between the inverting part and the Arduino that only takes positive voltage to avoid extra loading effect.

Since the voltage relationship given by the inverting op-amp part follows $V_{out}= -(R_{thermositor}/R)V_{pot}$, and the resistance for this thermistor was tested 121 kΩ near room temperature, we chose the resistor R of 100kΩ correspondingly to make sure the output voltage is less than V_{pot} as the temperature increases (the resistance of thermistor decreases, therefore the output voltage). The corresponding relation between the constants and the voltage now is $V_{out}= Ce^{B/T}$, which can also be written as $\ln V_{out} = \ln C + B/T$. Since the characteristic constants for individual thermistors differ even they are of the same type, in order to calibrate the setup, by wiring the

thermistor with a thermometer in a beaker of water being heated, we recorded several data points for the output voltage signals with the corresponding temperature, and graphed the relationship between $\ln V_{out}$ and 1/T to calculate the best-fit intercept that determines lnC (therefore the experimental value of C) and the slope that determines B (the experimental value of constant B).

After the calibration was done, with the known voltage-temperature relationship, we went on to build a temperature controller. In addition to the same thermistor that worked as the sensor part, we used a microcontroller (Arduino) to serve as the circuit controller connected to an immersion heater through a Solid-State Relay, a multi-outlet device used to receive the order and turn on or off the heater, shown in Figure 3. In order for maintaining a desired temperature, we designed the microcontroller such that it turned on the heater once it received the signal from the sensor that the temperature is below a certain value, and similarly it turns the heater off once a maximum is reached.

Experimental Setup - Software

The very first step in Arduino programming is to set up the pins; in this context, pin A0 was the input for the thermistor circuit, and pin 9 was the output for controlling immersion heaters. Followed is a loop function in which we had the program read the temperature automatically by the conversion: T = B/(log(inVolt/C)), where inVolt is the read from the analog input. The actual temperature controlling part in the loop function took advantage of *if* function - that is, if the temperature read is less than the minimum of the range to be maintained, the *digitalWrite* function will put the digital output in High to send an electric signal to turn on the heater, and if the temperature read is greater than the maximum, *digitalWrite* will turn to Low so that the

heater will be switched off. The detailed sketch with annotations is shown in Appendix 2 - Sketch 3.

Results and Discussion

By recording five temperature data points: 48, 50, 58, 63, and 68 in Celsius, and the output voltages: 1.55, 1.40, 1.03, 085, and 0.69 in Volts, correspondingly, we took the inverse of T's and the natural log of V_{out}'s, and made a $\ln(V_{out})$ vs. 1/T shown in Figure 4. By regression analysis, we found the slope for the best-fit line being 127.9381 with the uncertainty 6.5698, and the intercept value being -2.2143 with the uncertainty 0.1175. Since lnC is our intercept value, we had the parameter C = e^(-2.2143) = 0.1092, while parameter B is the slope 127.9381.

However, when we put our experimental parameter values into the Arduino sketch, shown in Appendix 2 - Sketch 3, as we set the temperature reading between 58 and 62, the immersion heater did not go on or off as expected. Since the thermometer read kept going up and displayed no trend of slowing down for a long while, along with the fact that the metal part of the heater kept bubbling, the main cause should not be the hot metal giving off the excessive heat due to the difference of heat capacitance. We changed the numbers that define the temperature range several times, and it was observed that when the temperature range was set between 15 and 16, the temperature was maintained around 60 degree Celsius. Therefore, the problem lies in the calibration parameters, and they are likely arising from previous calculation mistakes or the measurement inconsistency with the thermometer readings.

Summary

By taking advantage of the thermal sensitivity of a thermistor, we first built a calibration circuit to find its characteristic parameters that provided an analytical relationship between the temperature sensed by the thermistor and the circuit output voltage. We then programmed a microcontroller – Arduino to give order to the heater whose heating metal part was immersed in a water such that it turns off automatically when the temperature went above 60 degree Celsius and on when below. The controlling process was achieved, however, the calculated temperature value put in the program was not consistent with the real one. In order to avoid such inconsistency as much as possible, during the calibration process, one should make sure the thermistor sensing tip is tight with that of the thermometer, and record the voltage and the temperature reading as synchronous as possible.

Reference

[1] Tarun, A. *Stepper Motor – Types, Advantages & Applications.* Retrieved from https://www.elprocus.com/stepper-motor-types-advantages-applications/

[2] Clayhold, J., Jaeger, H., Pechan, M., Priest, J. (2016). *Electronic Instrumentation Laboratory.* Department of Physics, Miami University, Oxford, Ohio

Appendix 1 - Figures

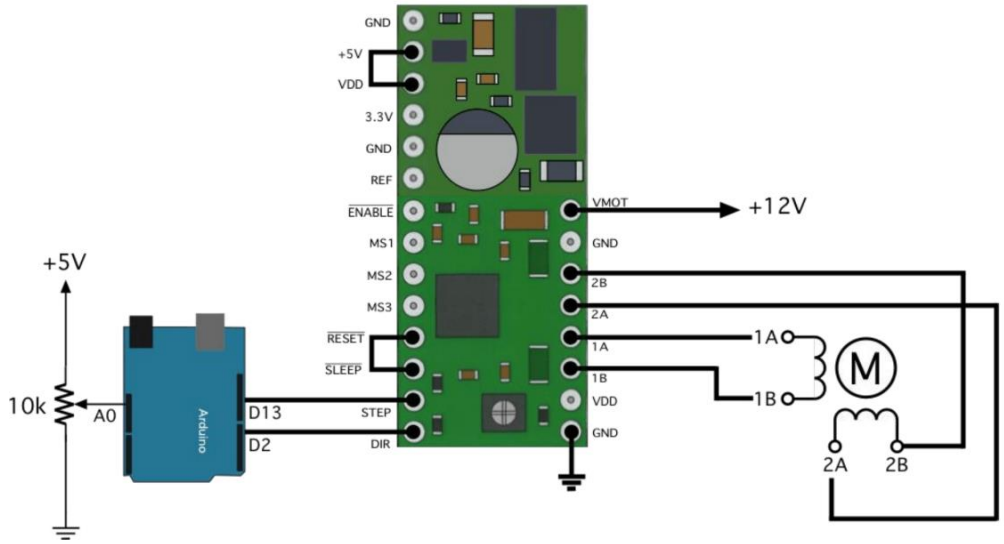

Figure 1: Stepper motor controlling circuit diagram. From PHY 294 Lab Manual, Lab set #7, used with permission from the authors. [2]

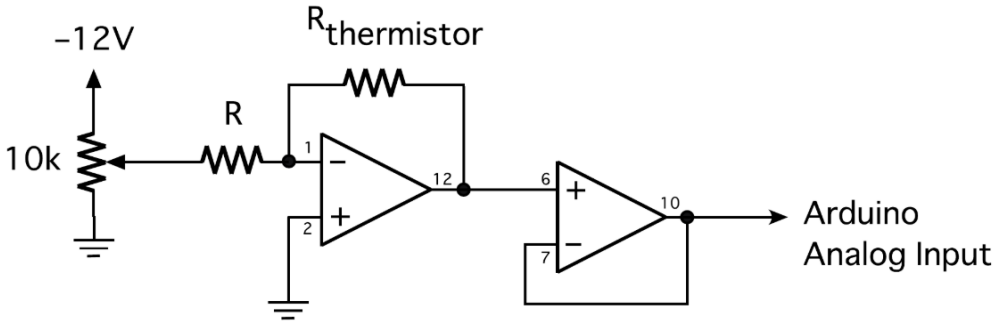

Figure 2: Temperature controlling calibration process circuit diagram. From PHY 294 Lab Manual, Lab set #4, used with permission from the authors. [2]

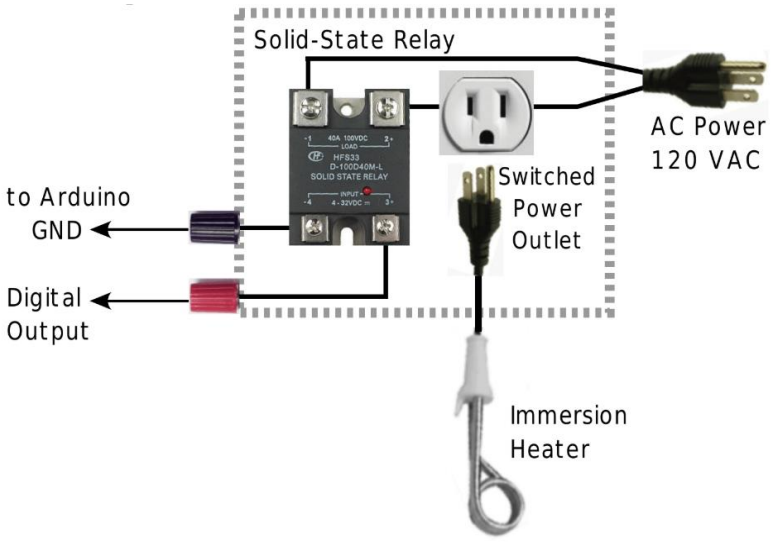

Figure 3: Solid State Relay with its outlet connections. From PHY 294 Lab Manual, Lab set #4, used with permission from the authors. [2]

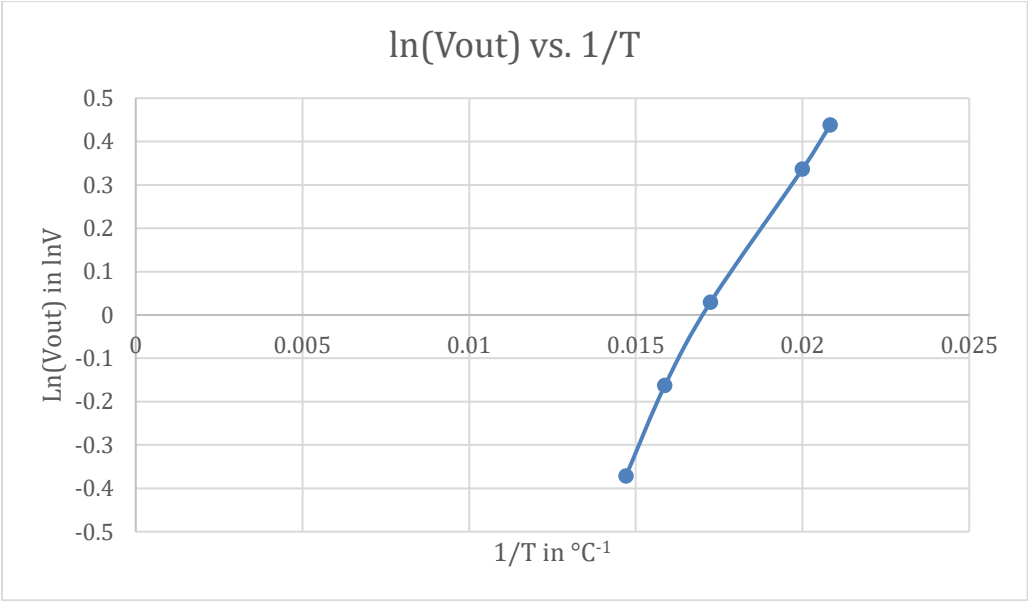

Figure 4: Graph for the measurement of ln(V_{out}) versus 1/T in the temperature controlling calibration part.

Appendix 2 - Sketches

Sketch 1: Step Size of the Motor

```
const int analogInPin = A0;          //Analog input for the pot
const int DirPin = 2;                //Digital output pin for direction
const int StepPin = 13;                    //Digital output pin for stepping
const int buttonPin = 6;                   //Digital input pin for button
int potValue = 0;                    //Value read from the pot
int buttonValue = 0;                       //Value read from the button
int count = 0;                       //counts of number of steps during revolution

void setup() {
  pinMode(DirPin, OUTPUT);
  digitalWrite(DirPin, HIGH);        //Forward = increasing pot voltage
  pinMode(StepPin, OUTPUT);
  digitalWrite(StepPin, LOW);
  pinMode(buttonPin, INPUT);
  Serial.begin(9600);
}

void step(int x) {
//==== one step followed by followed by a delay of x milliseconds
  digitalWrite(StepPin, HIGH);
  delay(x);
  digitalWrite(StepPin, LOW);
  delay(x);
}

void loop() {
  buttonValue = digitalRead(buttonPin);      //Reads if the button is pressed down or not
  while(buttonValue == 1) {          //while the button is pressed down it steps the motor
        step(10);
        count++;                                    //increases count every step
        buttonValue = digitalRead(buttonPin);       //Reads if the button was released every loop
  }
  if(count != 0) {                   //Assures that the serial monitor isn't spammed with 0's
        Serial.println(count);             //Prints the count
  }
  count = 0;                         //resets the count to 0 for further testing
```

```
}
```

Sketch 2: Backward-Forward Sketch

```
const int analogInPin = A0;    //Analog input for the pot
const int DirPin = 2;                  //Digital output pin for direction
const int StepPin = 13;                //Digital output pin for stepping
int potValue = 0;                      //Value read from the pot

void setup() {
  pinMode(DirPin, OUTPUT);
  digitalWrite(DirPin, HIGH);          //Forward = increasing pot voltage
  pinMode(StepPin, OUTPUT);
  digitalWrite(StepPin, LOW);
  Serial.begin(9600);
}

void step(int x) {
//==== one step followed by followed by a delay of x milliseconds
  digitalWrite(StepPin, HIGH);
  delay(x);
  digitalWrite(StepPin, LOW);
  delay(x);
}

void loop() {
  step(10);                            //Move stepper motor one step
  potValue = analogRead(analogInPin);        //Read the pot voltage
  Serial.println(potValue);
  if(potValue < 600) {                 //The analog value for 3V is 600
        digitalWrite(DirPin, HIGH);                 //change direction to forward when at 3V
        delay(3000);                                //wait 3 seconds
  }
  if(potValue > 800) {                 //the analog value for 4V is 800
        digitalWrite(DirPin, LOW);                  //change direction to reverse when at 4V
        delay(3000);                                 //wait for 3 seconds
  }
}
```

Sketch 3: Temperature control

```
const int analogIn = A0;                    //analog input for the thermistor circuit
const int digitalOut = 9;                   //output for controlling the immersion heaters

void setup() {
  pinMode(analogIn, INPUT);
  pinMode(digitalOut, OUTPUT);
}

void loop() {
  int inVolt = analogRead(analogIn);        //the voltage is read from the analog input
  temp = 127/(log(inVolt/0.109));           //that voltage is converted using this equation
  if(temp < 15)                             //turns immersion heater on if below 15 C
        digitalWrite(digitalOut, HIGH);
  if(temp > 16)                             //turns immersion heater off if above 16 C
        digitalWrite(digitalOut, LOW);
}
```