SANTA CLARA UNIVERSITY	Mechatronics 2024	Andy Wolfe

Lab #6 – Positioning Motors

I. Objectives

To study the operation and performance of Servo Motors and Stepper motors

II. Pre-Lab

Preparation:

- Read this lab.
- Review all of the documents posted in the parts folder related to the components in this lab and at the links in these documents (and the links from those links) so that you understand how the servo and the stepper motor work. Remember that I provided a component list at the beginning of the course.
- Figure out how to mount your servo and one of the distance sensors to your platform.
- Make sure you have your code from Lab 2.

Pre-Lab Report:

- Determine the maximum torque provided by the servo with a 4.8V power supply.
- Determine the maximum torque provided by the stepper motor with a 12V power supply at 200steps/s.
- Explain which motor has more torque.
- Peak torque in a Porsche Taycon Turbo S is 774 lb-ft.
  https://www.porsche.com/usa/models/taycan/taycan-models/taycan-turbo-s-cross-turismo/
  How many lab stepper motors would it take to equal the torque in that Porsche? (Pay attention to units).
- Include a selfie from your planning meeting including something not from lab that contains an electric motor.

III. Lab Procedure

You can perform the experiments in either order.

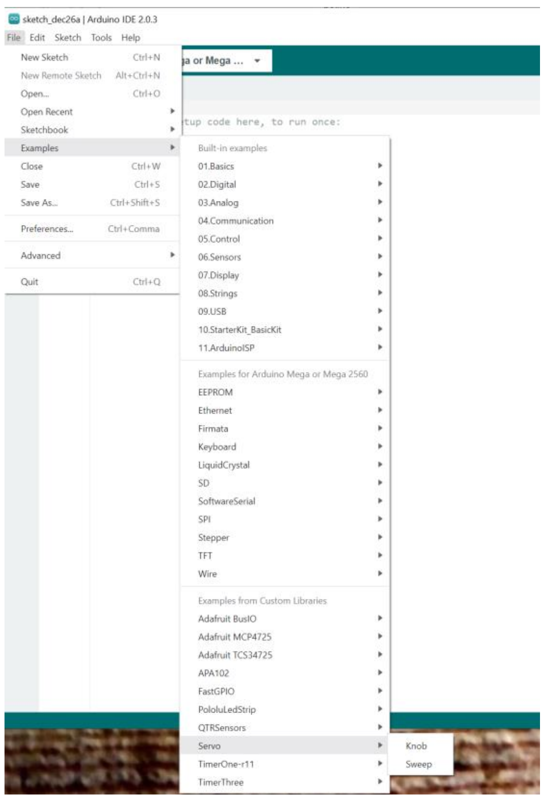
Experiment 1:

Servo Motors:

The RC servo motor is the most economical solution when positional control is required. Internally the RC servos have a DC motor (brushed or brushless), driver, controller with compensator inside. All it requires is DC power and a PWM control signal. There are some limitations to these simple servos. The price grows when more torque is required.

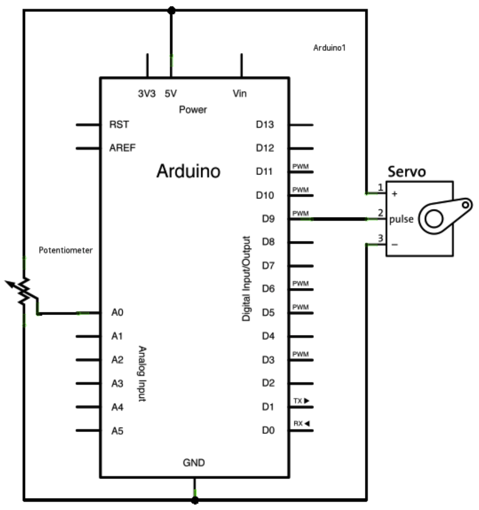
Procedure

1. Load program called Knob as shown in the following figure.



2. Connect the Arduino according to the schematic as shown below.
3. Connect the potentiometer center terminal to pin A0. The left pin to 5V and right pin to ground. Data sheet is online for Bourns 3362U-1-104LF.

4. Connect 5V and ground to the Servo: the red and black wires respectively. Connect the control signal (not the red and black wire) to digital PWM pin 9. (Some servos have an extra wire not on the 3-pin connector. Ignore it)



5. Turn the potentiometer to the center. What voltage is on pin A0? (answer in report)

6. Record the output waveform from the Arduino to the servo using the oscilloscope.

7. The servo control library uses the method servo.write(angle) to set the servo to a specific rotation. https://www.arduino.cc/reference/en/libraries/servo/write/

8. The servo might not be able to follow when you try and control the pulse-width of the control waveform to the far extremes. For the given range of pulse widths in the Arduino code, some models may travel 180° while some may not. Therefore a parameter value of 180 degrees to the servo.write(angle) method might not give you 180° of travel. Look at the specifications for our servo – it is rated at 120° of travel. When you reach the end of travel, the servo will shake very hard if you attempt to go beyond that point. **Do not drive the servo beyond end of travel for more than a few seconds since you may damage it.**

9. Find the voltage of analog input A0 such that the servo just reaches the end of travel on each side. What are the pulse durations of the waveform corresponding to the end of travel on each side? Record those waveforms using the oscilloscope (phone picture or hand drawing is ok).

10. Modify the Arduino program to determine the angle parameter value that should be provided to servo.write(angle) that corresponds to the end of travel for each side. **Modify Knob so that the servo travels through its full range – but never hits the limiters at the extremes**. Turn that code in with your report.

11. As you try to do more with servos, remember to be aware that it is possible that the servo library grabs timers on the Arduino. I believe that the current library version does not but be aware.

Now, mount one of your distance sensors (Sharp or homemade) on your robot. Mount one of the arms on the servo motor and mount a piece of cardstock or cardboard behind the arm. Using code from Lab 2, measure the distance to a wall from 5-40cm. Draw a dial on the cardboard and control the servo motor so that it acts as a distance meter and points to the correct distance.

(Demo #1) Show it working.

Experiment 2:

The servo motor has some limitations. It cannot continuously rotate. It cannot even rotate 180 degrees. This is where the stepper motor can help you. Stepper motors generally use an open loop control to control the position. It doesn't know where it is. The only control you have is how many steps you want to go and which direction. (You can add a sensor to a stepper motor for closed loop control, but it is not built in.) As long as you keep track of where you are and the motor never slips, the stepper can be quite precise. Be aware that when you power down, there is no way to make sure that the motor does not slip or keep track of where you are.

A stepper motor can be controlled with stepper motor driver. You have been supplied with a stepper motor driver board called EasyDriver. (It should be version 4.4 or 4.5 – see http://www.schmalzhaus.com/EasyDriver/, http://www.schmalzhaus.com/EasyDriver/Examples/EasyDriverExamples.html, and https://www.sparkfun.com/products/10267.) It requires 2 control signals: DIR for direction and STEP for the step control signal.

Stepper motors requires you to keep track of how many steps are needed in order to turn the stepper motor. Open loop control relies on the user to keep track of the motor location. The EasyDriver board supports Microstepping (http://en.wikipedia.org/wiki/Stepper_motor#Microstepping). The stepper motor is 48 steps per revolution and the stepper controller being used in lab should microstep at 8x that rate to take 384 steps to turn the stepper motor one revolution. This means that if you attach a wheel to a stepper motor, the distance travelled per step = $\mathbf{2\pi R/384}$ where R is the radius of the wheel you attached to the stepper motor. You can use this to determine number of steps needed to travel a given distance or vice versa.

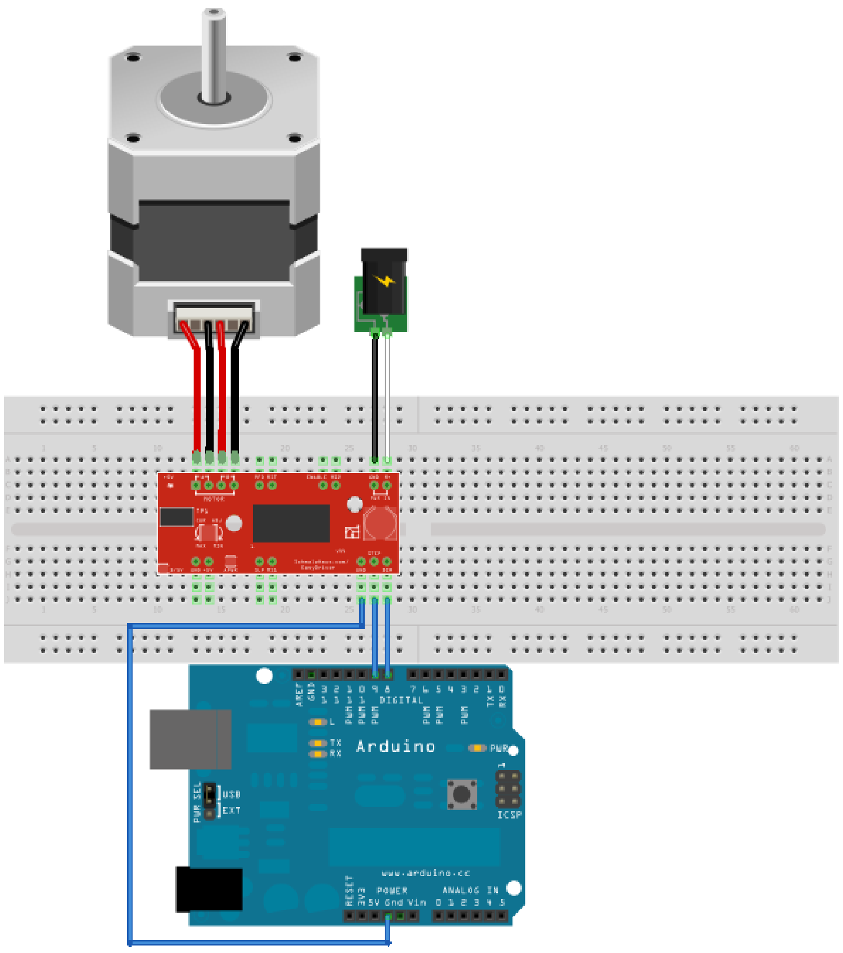
The chips on the stepper motor controller board get very hot in operation – they will burn you if you touch them! Also – be careful to watch the proto board. If there is any sign of melting, stop and get help.

Procedure

1. Plug the stepper motor driver into the proto board and connect as shown below. Use a lab bench power supply set to 6V and 50% current limit to power the stepper motor board. The stepper motor documents on the Google Drive identify the wire colors.
2. Plug the **DIR** pin of the driver board to pin 8 of the Arduino and the **STEP** input to pin 9 of the Arduino.
3. Run the following program: Your motor should run.

```
void setup() {
  pinMode(8, OUTPUT);
  pinMode(9, OUTPUT);
  digitalWrite(8, LOW);
  digitalWrite(9, LOW);
}

void loop() {
  digitalWrite(9, HIGH);
  delay(1);
  digitalWrite(9, LOW);
  delay(1);
}
```



Assignment

1. Record the waveform between motor terminals A+ and A- to ground using DC coupling on the scope.
2. Explain why the waveform is so weird.
3. Modify the program to run the motor 384 microsteps forward, wait 1 second, then 384 microsteps backwards. Place some tape on the shaft and measure the amount of rotation that corresponds to 384 microsteps. Verify or disprove my theory that the motor requires 384 microsteps per revolution. (Demo #2 – show the motor running this program)