Experiment M4
Stepper Motors
Procedure

Deliverables: Checked score sheet, motor mount with holes drilled

Overview

Background

A stepper motor is very different from the previous DC motors. It uses multiple coils of wire that are sequentially switched ON and OFF by a *stepper driver* circuit board. A permanent multi-pole magnet is fixed around the motors shaft. Turing a coil ON will cause the multi-pole magnet to rotate and align a magnetic pole with the coil. Turning that coil OFF and its neighboring coil ON causes the magnetic pole to rotate to align with the neighboring coil.

In this lab you will learn to control a stepper motor and integrate it into a belt-drive system to create a linear actuator. Illustrated in Figure 1, the stepper motor will turn a pulley that will move a belt and gantry cart to a desired position *x*. Limit switches on the ends will prevent the motor from crashing the gantry cart into the ends of the track.

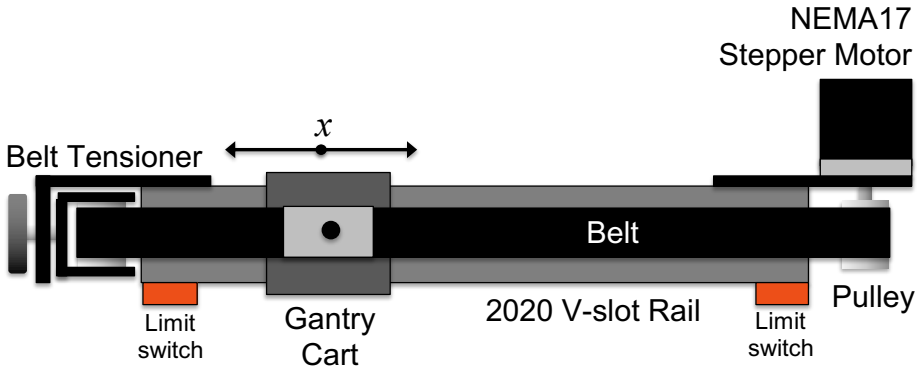

Figure 1 – A linear actuator system uses a stepper motor with a pulley and belt to move a gantry cart to a desired position *x*.

Part I: Stepper Motor and Driver Board

You will begin by connecting the motor to the stepper driver board programing it to spin to various angular positions.

Procedure

1. The stepper motor is a standard "NEMA 17" stepper motor, which means the motor case is 1.7" wide. This motor has 200 steps per revolution. Write this down in your lab notebook.
2. This particular stepper motor has two sets of coils. Each coil has two wires connected to it. Determine which pairs of wires are connected to a given coil. Use the handheld DMM to measure the resistance between two wires. A coil will have a resistance less than 50 Ohms.

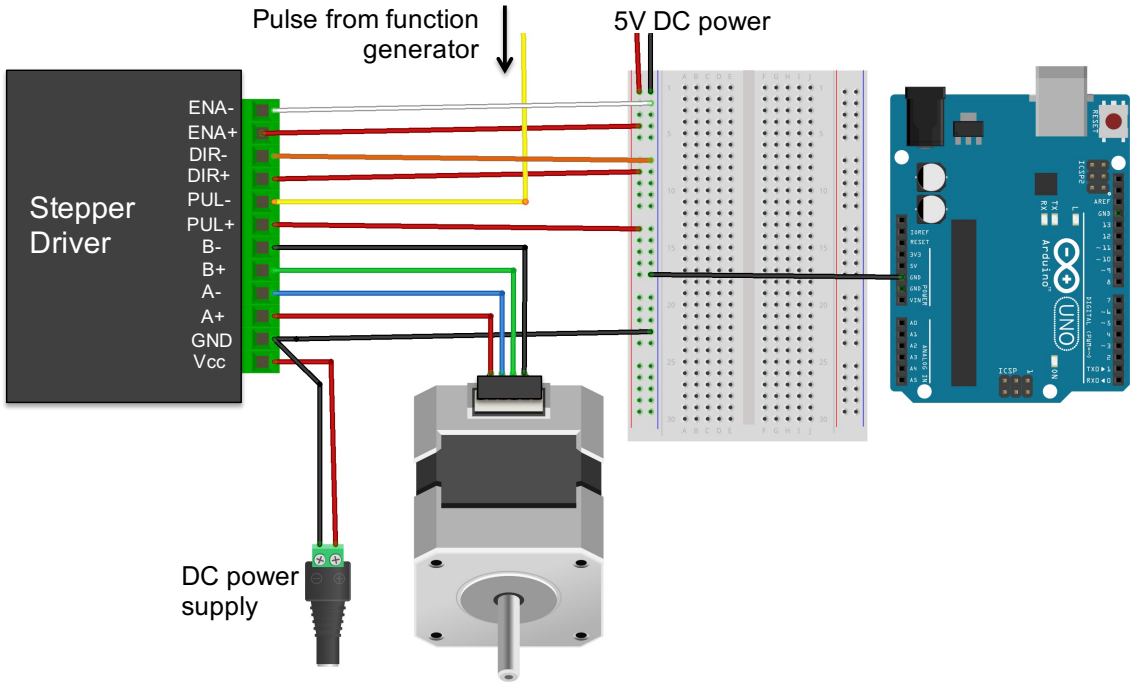

Figure 2 – The stepper motor, stepper driver, and Arduino microcontroller are all connected to a breadboard. A 12V power supply will power the motor coils and a 5V power supply is used to power the digital logic.

3. Write down which two color wires share a coil in your lab notebook. Connect each pair of motor wires to the A and B terminals of the stepper driver, as shown in Fig. 2.
4. Wire up the + and – bus lines on the large breadboard to its +5V power supply and ground.
5. Connect the following pins on the stepper driver to +5V on the breadboard: ENA+, DIR+, and PUL+.
6. Connect the following pins on the stepper driver to ground on the breadboard: ENA- and DIR-.
7. Locate the 12V DC power supply. (It looks like a laptop charger.) Make sure the 12V DC power supply is unplugged from the AC wall outlet.
    a. Connect the kill switch and green screw terminal adapter, as you did in previous labs.
    b. Connect the + terminal of the power supply to the VCC screw terminal on the stepper driver.
    c. Connect the – terminal of the power supply to GND on the stepper driver.
8. Use a male pin jumper wire to connect the GND screw terminal on the stepper driver to ground on the breadboard. (You will have two wires screwed into GND on the stepper driver.)
9. Connect ENA- to +5V on the breadboard to enable the stepper. (Connecting it to GND (LOW) will disable the stepper driver. This is useful to conserve power and prevent the motor coil from becoming too hot.)

10. Use the function generator to drive the PUL- (pulse) pin. Remember, the black minigrabber always goes to ground. Start with a 600 Hz "Pulse" with a high value of 5V and low value of 0V.
11. Put a piece of masking tape on the shaft of the stepper motor so you can see it spin.
12. Plug the 12V DC power supply into the wall outlet. Turn on the output from the function generator. The shaft should begin to rotate.
13. Try different frequencies between 100 and 1600 Hz. Which frequency results in the smoothest operation of the stepper motor? Why do you think it is choppy at low and high frequencies?
14. Change the DIR- (direction) pin from ground to 5V on the breadboard. This should change the direction of rotation. **Demonstrate the working system to the lab instructor to receive credit on your score sheet.**
15. Disconnect the function generator and turn it off. Turn off the 12V DC power supply. Leave everything else connected to the breadboard.

Part II: Microcontroller Implementation

You will now use the Arduino UNO microcontroller to generate the step pulses and control the direction.

Procedure

1. Power down the entire system and make the following connections with the Arduino:
   a. Arduino GND (any of them) → grounded bus line on the breadboard
   b. Arduino Digital Pin 7 → ENA- on stepper driver
   c. Arduino Digital Pin 8 → DIR- on stepper driver
   d. Arduino Digital Pin 9 → PUL- on stepper driver
2. Download the Stepper Driver Example code from the M4 webpage. Examine the code and try to understand what it will do.
3. Run the sketch on the Arduino. You should see the stepper motor spin.
4. Modify the code to make the stepper spin in the opposite direction. (Change the value assigned to the DIR- pin.)
5. Try adjusting the microsecond delay time in the delayMicroseconds() function. Find a time that results in the smoothest motion with minimum vibration.

Pro-tip: Print statements are useful for debugging code. However, the Arduino must pause to send this information to the computer. Too many print statements may cause the system to become choppy. Try commenting out print statements if the stepper motor exhibits excessive vibrations.

6. Write a program that will repeatedly rotate the shaft one full revolution clockwise, then one full rotation counter-clockwise. **Demonstrate the working system to the lab instructor to receive credit.**
7. Save all of your codes to the M4 folder in your code library.

Part III: Mechanical Assembly

You will now assemble the linear belt drive system, which will use pulleys to convert the rotational motion to linear translational motion.

Procedure

1. Power down the system. Turn off an unplug the 12V DC power supply. Turn off the breadboard. Unplug the USB from the computer.
2. Mount the stepper motor to the NEMA 17 mounting plate using the M3 screws and round spacers. Use the round spacers to create a gap between the stepper and the mounting plate.
3. Mount the pulley to the shaft of the stepper motor. Tighten the set screw onto the flat part of the shaft. Make sure the pulley does not rub against the motor.
4. Use M4 T-nuts and screws to mount the motor plate to the 2020 slotted rail. (The M4 screws and T-nuts have a 4 mm diameter—slightly bigger than the M3 ones.) The pulley should be in line with the center of the rail.
5. Slide the gantry cart onto the rail, such that the wheels run in v-slots on the rail.
6. Mount the belt tensioner to the opposite end of the rail using M4 T-nuts and screws.
7. Turn the knob on the belt tensioner to move the pulley closer to the rail.
8. Run the timing belt through the pulleys. The belt teeth should interlock with the pulley teeth.
9. Loop the belt ends through the top of the slots on the gantry cart. Use small zip ties to clamp the belt to itself, such the belt teeth interlock with each other. Tighten the zip ties and cut off the tag ends. (See the demo set-up.)
10. Gently tighten the belt using the knob on the belt tensioner.
11. Use your hand to manually move the gantry cart back and forth. It should move fairly easily.
12. Power up the system, and re-run your code from Part II. (Keep your hand on the kill switch so you can quickly turn it off if it crashes.) The stepper motor should drive the cart back and forth.
13. **Demonstrate the working system to the lab instructor to receive credit.**

Part IV: Actuator Calibration

You will now calibrate the system and determine the calibration constant a in units of steps/cm.

Procedure

1. Modify the original example code to move the cart 1000 pulses to the left, then stop and disable the motor.
2. Measure how far the cart travels to the left in 1000 pulses. (Keep your hand on the kill switch so you can quickly turn it off if it crashes.)
3. Modify the code to move the cart 1000 pulses to the right, then stop and disable the motor.
4. Measure how far the cart travels to the right in 1000 pulses.

5. Record these values in your lab notebook. Also note which bit value (HIGH or LOW) corresponds to which direction (left or right).
6. Use your measured values to compute the calibration constant *a* in units of steps/cm. **Show the calibration constant to the TA to receive credit.**

Part V: Limit Switches

You will now add single-pole double-throw (SPDT) limit switches to the system. The limit switches will disable the motor before it crashes into the ends of the track. They *limit* the motion of the cart.

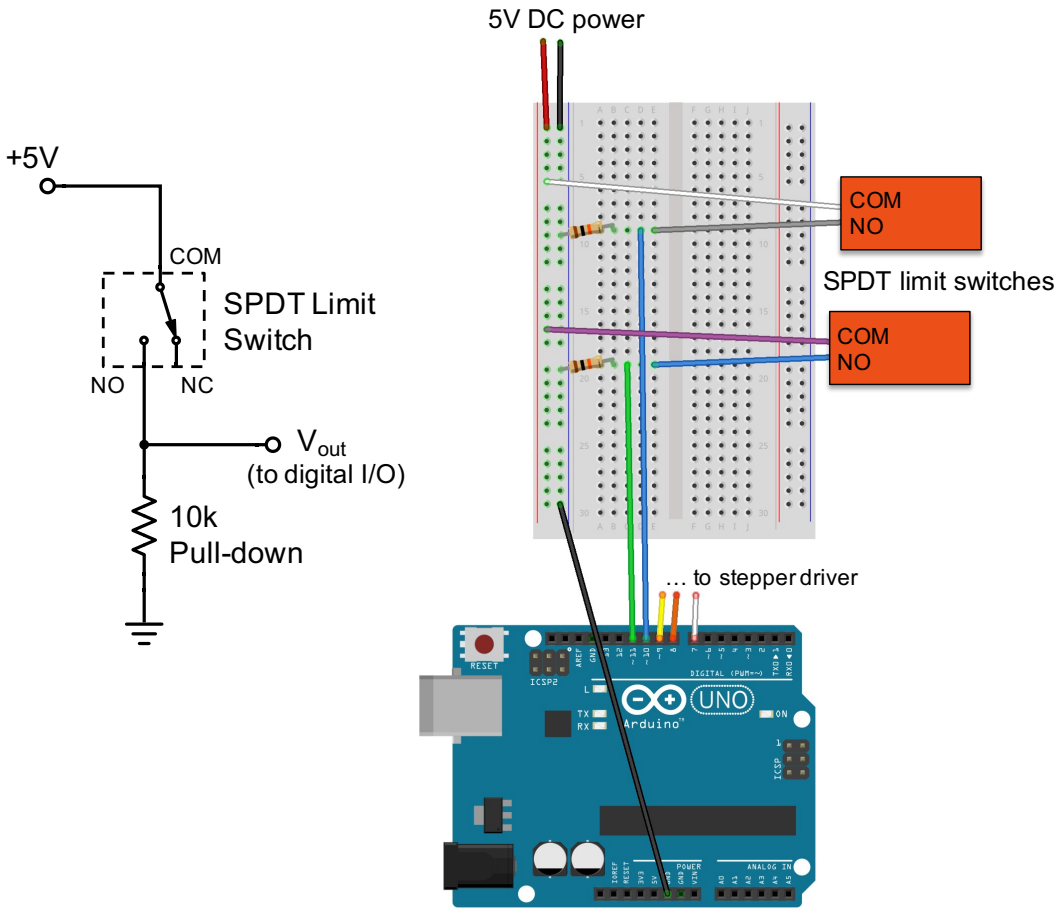

Figure 3 – The limit switches are connected in series with a 10k pull-down resistor. The output voltage across the 10k resistor is read by digital pins 10 and 11 on the Arduino. (The stepper motor connections have been omitted for clarity.)

Procedure

1. Power down the system. Turn off an unplug the 12V DC power supply. Turn off the breadboard. Unplug the USB from the computer.
2. Locate the limit switches and test them. Measure the resistance between the two wires. Pressing the lever arm should make a 'click' sound, and the resistance should change.

3. Use M3 T-nuts and screws to mount the limit switches to the sides of the rail, as illustrated in Fig. 1. (Note that the M3 T-nuts have a slightly smaller inner diameter than the M4 T-nuts.) The M3 T-nuts can be "dropped in" to the slot on the side with the rounded side facing inward. The limit switches should then be positioned at a height where the cart will depress the lever arm before crashing.
4. Connect the wires on the limit switches to the breadboard, as illustrated in Fig. 3.
5. Power up the circuit with 5V DC, and measure the voltage across the 10k pull-down resistor. It should normally be at 0V (LOW). Pressing the lever arm on the switch should drop the voltage down to 5V (HIGH).
6. Connect the output of the left limit switch to digital Pin 10, as illustrated in Fig. 3.
7. Connect the output of the right limit switch to digital Pin 11, as illustrated in Fig. 3.
8. Modify your code from Part III to contain a conditional 'if' statement that will disable the stepper if either limit switch is pressed. You will need to define pins 10 and 11 in to be inputs in the "setup" portion of the code.

Note: A variable x can be *inverted* from HIGH to LOW and vice versa using the syntax !x.

9. Power up the system and test the program with the limit switches. As soon as the motor starts moving, use your hand to press one of the limit switches, and the motor should stop. Perform this test on both limit switches.
10. Modify the code so the cart will intentionally crash into the limit switch. The limit switch should cause the microcontroller to disable the motor. Perform this test on both limit switches. **Demonstrate the working system to the lab instructor to receive credit.**

Part VI: Design Challenge 1 – Auto-calibration or "Touch-off"

You will now program the system to automatically calibrate itself or "touch-off" the ends by counting the number of steps between the limit switches. Using the known distance between the limit switch, the system can count the number of steps it takes to traverse this distance, then compute the calibration constant a in units of steps/cm.

Procedure

1. Turn off the 12V DC power supply.
2. Measure the distance between the lever arms of the two limit switches d. Record the value.
3. Download the Part VI Template code from the M4 webpage. Open the code in the Arduino IDE software.
4. The code contains a sequence of while loops nested in the main loop.
   a. The first loop should move the cart to the left limit switch.
   b. The second loop should move the cart to the right limit switch, then compute the calibration constant a in units of steps/cm.
   c. The third loop should move the cart to the left, stop in the center, and disable the stepper. We say the cart is now "resting in the home position".

d. The fourth loop is an infinite loop that simply prints the calibration constant *a*, while the cart rests in its central home position.

5. Modify the code and replace the "***" with the correct values, parameters, and formulas indicated by the comments.
6. Turn on the 12V DC power supply and test the code. (Keep your hand on the kill switch in case the cart crashes.) If it crashes, manually move it back to the home position and think about what you did wrong.
7. **Demonstrate the working system to the lab instructor to receive credit.**

Part VII: Design Challenge 2 – Position Control

Time for the grand finale. After performing the auto-calibration procedure, the system should prompt the user to specify a position *x* along the linear rail. The coordinates will be defined such that the left limit switch is at $x = 0$ and the right limit switch is at $x = d$.

Procedure

1. Turn off the 12V DC power supply.
2. Save a new copy of your code from Part VI with a new intelligent file.
3. Modify the code to include a variable x_{SET} that represents the desired position.
4. Define two new functions at the end of the code, after the main loop:
   a. void moveLeft(float distance){} should move the cart to the left by an amount equal to the specified 'distance' in cm.
   b. void moveRight(float distance){} should move the cart to the right by an amount equal to the specified 'distance' in cm.
   c. Both function should set the direction pin to the appropriate logic (HIGH or LOW), enable the stepper driver, use the calibration constant *a* to determine the number of pulses, then enter a loop to create this number of pulses. After the movement is complete, the enable pin should be set to LOW to disable the stepper.
5. Modify the fourth while loop (the infinite loop) to prompt the user to enter a position x_{SET} in cm, then move to that position.
   a. See the following example: https://www.arduino.cc/reference/en/language/functions/communication/serial/read/
   b. Use Serial.println() to ask the user to specify the position in cm.
   c. Use Serial.parseFloat() (instead of Serial.read) to read the position entered by the user.
   d. Check that the position will not cause the cart to crash. (Don't forget to account for the length of the cart.) Print a warning message if the value is out of bounds.
   e. Compute the distance the cart will need to move to either the left or right to obtain the new position *x*.

f. Use an 'if' conditional to call the appropriate function moveLeft() or moveRight() to move the cart to the desired position.

g. After the move is complete, update the position to be $x = x_{SET}$.

6. **Demonstrate the working system to the lab instructor to receive credit.**

Clean-up

To receive full credit, you must return the lab bench to its initial state:

- Unplug the DC power supply. Remove the kill switch and screw terminal adapter.
- Disassemble the circuit. Disconnect the wires from the stepper driver, Arduino, and breadboard.
- Disassemble linear actuator.
- Put anything that belongs in your tool kit back into the tote bin.

Data Analysis and Deliverables

You do NOT have to write a tech memo for this lab. Rather, you have to go to the machine shop and create a mount for the solenoid valve.

Appendix A

Equipment

- 12V DC power supply (looks like a laptop charger)
- BNC to minigrabber cable
- Large breadboard with 5V DC power supply
- Arduino Microcontroller
- Microstep stepper driver (Amazon Part # B07RRB6BGQ)
- NEMA 17 stepper motor (42A02C)
- Ribbon cable for NEMA 17 motor
- NEMA 17 mounting plate
- Pulley
- 2020 v-slot rail, 50 cm long
- V-slot gantry cart
- Belt tensioner for 2020 rail mount (Amazon Part # B07XL41QNH)
- Timing belt, 5 mm wide, 105 cm long
- Small zip ties
- M4 T-nuts and screws
- M3 T-nuts and screws
- 2 Limit switches (1 w/ white and grey wires, 1 w/ purple and blue wires)
- 8020 bracket for making motor mount (deliverable for following week)