Stepper Motor Lab

Learning Objectives:

After successfully completing this lab, students will be able to:

- Describe the advantages and disadvantages of stepper motors vs. DC motors
- Describe the difference between unipolar and bipolar stepper construction and full-step drive control
- Describe the advantages and methods of regular vs. high-torque control
- Describe the methods of micro-stepping control
- Use the provided Arduino Stepper library for basic stepper control
- Program the Arduino to perform full-step and microstep control
- Program the Arduino to perform servo-style absolute position control using half (or finer) step control
- Utilize serial communication to the Arduino from the serial monitor

Components:

Qty.	Item
1	Arduino microcontroller board and USB cable
1	six-wire stepper motor with laser-cut position indicator wheel
1	L298-based motor driver board
1	Potentiometer

Introduction

In this lab you will explore the types of stepper motors and their variety of construction, operation, and control. In lecture you have already learned about the theory of stepper motor construction and basic methods of control. With the exception of the variable reluctance stepper motor, stepper motors are constructed with permanent magnets in their rotors and electromagnetic drive coils with their surrounding stator to generate motion (torque). By energizing the coils in the proper order, the stepper can be made to move in small increments. These discrete, incremental steps occur due to the physical construction of the motor. The number of 'pole teeth' on the stator poles and the rotor determine the (angular) step size of a particular motor. You will now take a generic stepper motor and learn how to put it to use with the Arduino using a motor driver board.

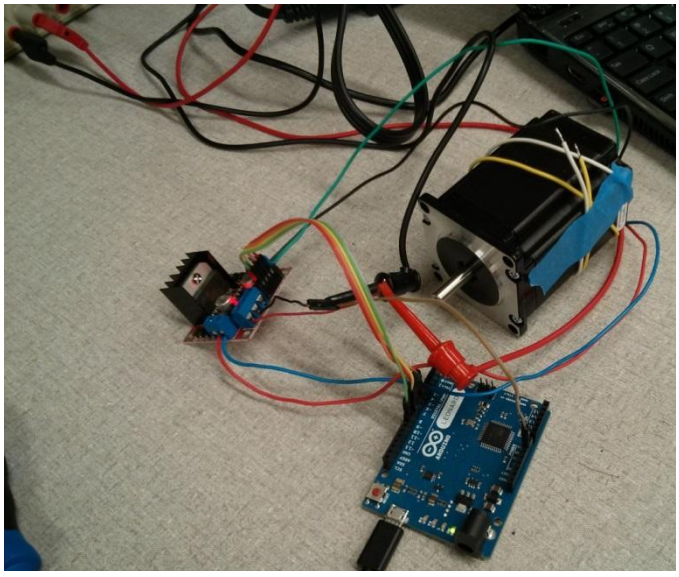

Figure 1. The stepper motor drive board connected to the Arduino.

Stepper Motor Wiring Configurations

The first task to perform when handed a (or when comparing multiple, random) stepper motors is to determine its wiring topology. There are two basic types of stepper motors – *unipolar* and *bipolar*. They are the same in their physical construction, but differ in the way that their coils are wired. Bipolar motors provide independent access to both ends of the drive coils, whereas unipolar steppers have one end of each coil tied to one (or more) common connection(s). An easy way to remember which construction is which is to remember that current in a unipolar construction is uni-directional, whereas in a bipolar stepper, current can flow in both directions, just like with a DC motor. An illustration of the wiring diagrams for each is shown in Figure 2.

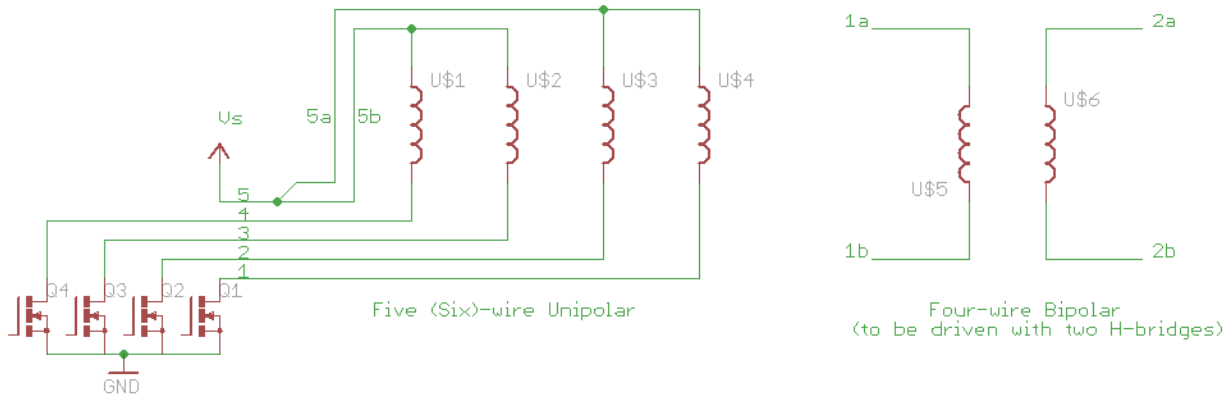

Figure 2. Unipolar vs. bipolar stepper wiring configuration. Note the five or six-wire (depending on whether or not the coils' center-taps are joined) topology of a unipolar vs. the four-wire bipolar stepper motor. More information on wiring topologies can be found on the Oriental Motors website, http://www.orientalmotor.com/stepper-motors/technology/unipolar-bipolar-connections.html

One can usually determine the type of winding configuration by counting the number of wires coming out of the motor body. As mentioned previously, unipolar steppers contain one or more common connections that are tied together inside of the motor body ("center taps"). If there is a single common connection (both coils' center-taps tied together internally), then five wires will be present in the wiring harness, otherwise if there are two common connections (center-taps independent), then six wires will be present. A bipolar stepper will contain four wires, one for each end of both coils. **(Lab-Report Q1) A unipolar stepper can be configured as a bipolar stepper. How would this be achieved?** For completeness, there is also a *universal* stepper motor which is a unipolar-bipolar hybrid and contains four independent coils, and therefore eight leads. A universal stepper can be converted to a unipolar stepper or a bipolar stepper by connecting their coils in parallel or in series, respectively[1].

Erics Solution…. A unipolar stepper motor can be configured as a bipolar stepper motor by disconnecting the center tap connections, i.e. do NOT connect the center tap connections to a common source. Therefore, the first set of coils (red and blue wires of the stepper) will be connected to each other, but not connected to the other set of coils (green and black wires of the stepper). This allows the entire coil to be used as a bipolar stepper. The center tap wires on our stepper motors in the lab are the yellow and white wires. So, we will leave them unconnected.

(Lab-Report Q2) In your lab report, include a table which one could use to identify a stepper motor based on the number of wires.

Erics….. Solution

Number of Wires	Type of Stepper Motor
4	Bipolar

[1] See http://www.orientalmotor.com/stepper-motors/technology/unipolar-bipolar-connections.html for more information.

6	Unipolar
8	Universal

One of the advantages of unipolar motors is that because current flows uni-directionally through the windings, a single transistor can be used to energize each coil. In contrast, with a bipolar motor, current needs to be made to flow in both directions, so an H-bridge (four transistors) is required for each winding. The advantage of a bipolar stepper is that it has a larger torque-to-size ratio in comparison to a unipolar stepper. This advantage might justify the more complicated drive requirement over the simpler unipolar configuration.

Stepper Motor Parameters

Besides their wiring configuration, stepper motors are characterized by their step geometries and the current rating of their coils. Step configurations may range between 0.72 and 90.0 degrees per step, and typical stepper motors are constructed to be in the 0.9/1.8/3.6/7.5/15.0-degrees per step range. If the simplest drive methods are used, the angular resolution of the stepper motor will be the basic step size. However, you will learn later that finer steps are possible by complicating the drive electronics.

The other important parameter is the maximum current that should be used in the motor's coil. Fairly often this parameter is accompanied by a DC voltage rating for the motor, which is often specified to be 12V or 24V. By using Ohm's law, one can determine the DC resistance of the copper coil winding given this voltage and current specification. This voltage is not the maximum voltage that one may use to drive the coil because the coils themselves are (quite) inductive, so a driving voltage may be substantially (2x-10x) larger than the rated DC voltage specification. It is the responsibility of the drive electronics, however, to ensure that the current at any point in time is chopped (limited) to the motor's coils' maximum current rating.

Every stepper motor has some physical parameters (rotor mass, magnet strength and tolerances, etc.) that, along with the character of the load and the implementation of the driver, determine the maximum step rate and resonant frequencies of a given stepper motor system. Note that there is a maximum step rate for this (or any) motor, above which the stepper will miss steps (or perform relatively badly, if operated at a resonant frequency).

Remember that it is always the case that the torque output of a motor is a function of the driving current, and not the voltage, as per the physics of electromagnetism. Voltage is the means by which current is produced in the coil wiring, and may be time-varying because of the inductance (and capacitance) of the drive and motor circuitry.

The datasheet for the stepper motor that we are using for this lab can be found in **Appendix A**, it has 200 steps per rotation, so 1.8° per basic step.

Exercise 1: Setup and Test

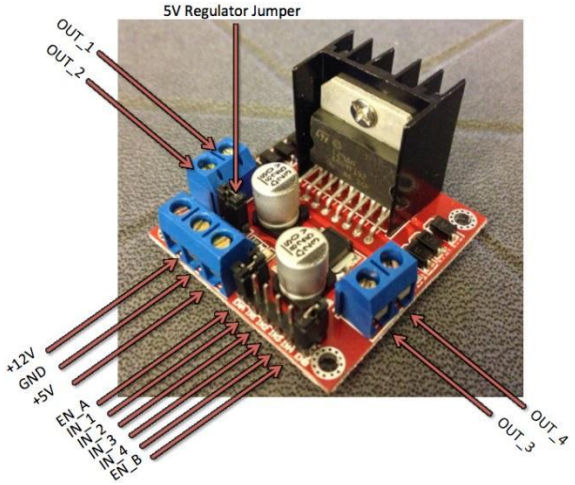

Figure 3. L298N Motor Drive Board

1. Upload the example sketch, *BareMinimum* included in the Arduino IDE to the Arduino board, located in File > Examples > Basic > BareMinimum. This is so that when you attach your motor driver, you will be sure that you are not telling it to do something you don't want it to be doing.
2. Be sure that the jumpers are present on the 5V regulator, AND the enable (EN_A and EN_B) headers (refer to Figure 3 if you are unsure where they are located). Skip to step 4 if your jumpers are present.
3. If the jumpers are not present, ask your lab instructor if they are available. Otherwise, follow a procedure similar to the previous servo lab to connect the 5V power and enable pins from your Arduino.
4. We will be connecting this stepper motor in a bipolar series configuration (**Appendix B**).
   - 4.1. The white and yellow center tap wires from the stepper motor will not be connected to anything.
   - 4.2. Connect the red and blue wires from the stepper to OUT_1 and OUT_2 respectively.
   - 4.3. Connect the green and black wires from the stepper to OUT_3 and OUT_4 respectively.
   - 4.4. Connect IN_1, IN_2, IN_3 and IN_4 on the driver board to digital pins 8, 9, 10, and 11 on the Arduino respectively.
   - 4.5. Now we need the power connections. Set your +20V power supply to 12V, then complete the following sub-steps **with the Arduino and power supply powered off**.
     - 4.5.1. Run a wire from the +12V terminal of the motor drive board to the +20V power supply terminal.
     - 4.5.2. Run another wire from the GND terminal of the motor driver board to the COM terminal of the power supply.
     - 4.5.3. Connect the Arduino's GND to motor driver board's GND terminal either by running a wire into the terminal or by jumping them together on a breadboard.
   - **4.6. Have the lab instructor approve your setup before proceeding to power up the power supply and Arduino.**

Now we need to determine the stepping sequence required to drive this stepper motor either clockwise or counter-clockwise. Upload the code located in **Appendix C** to the Arduino. After successfully uploading the code to your Arduino, open up the Serial Monitor terminal on the Arduino IDE. It may initially print out "Not a valid case." In the Serial Monitor terminal, type in either 1, 2, 3, or 4 and click the "Send" button. Each of these four numbers corresponds to their respective IN pins which are connected to your Arduino. (**(Lab-Report Q3)** Look through the code and comments for further explanation on how this code works and what it does).

(Lab-Report Q3) ….Eric Solution

The code from Appendix C uses the Serial functions (Serial.begin, Serial.available, Serial.read, and Serial.println) to take in user input from the Serial Monitor. The user is supposed to input different combinations of 1, 2, 3, and 4 on the Serial Monitor to determine the coil sequence that will move the motor CW or CCW. Each stepper motor will have a different coil sequence.

To summarize, focusing on the loop( ), the code from Appendix C does:

1. The program waits for user input from the Serial Monitor using Serial.available() > 0. If user input is detected, Serial.available() will be greater than 0, and the if-statement will be triggered.
   a. The combination that the user inputs will be read using Serial.read() and be stored into a variable called serialData, which is an int-type
   b. Also, it will print the combination that the user inputted using Serial.println
2. Following the if-statement, the serialData variable will be passed to a switch-case

a. Each number in the combination that the user inputted will be evaluated by each of the 4 case statements, and activate 1 of the 4 corresponding IN_ pins on the H-bridge, in the order that the user inputted.

Code from Appendix C:

```
// define the connections to motor driver inputs
#define IN_1 8
#define IN_2 9
#define IN_3 10
#define IN_4 11
// define the delay time for steps
#define DELAY_TIME 30

int serialData = 0;

void setup()
{
  // begin serial communication
  Serial.begin(9600);
  // setup the pins to motor driver as outputs
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);
}

void loop()
{
  // look for when data is available for read from the serial monitor console
  if (Serial.available() > 0)
  {
    serialData = Serial.read();  // read in the the serial data to a variable
    Serial.println(serialData);  // also print out serial data entered
  }

  // use variable as a switch in which the ASCII characters 1 through 4 correponds to
each of the pins
  switch (serialData)
  {
  case 49: // ASCII  value for '1'
    digitalWrite(IN_1, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_1, LOW);
    break;
  case 50: // ASCII  value for '2'
    digitalWrite(IN_2, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_2, LOW);
    break;
  case 51: // ASCII  value for '3'
    digitalWrite(IN_3, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_3, LOW);
    break;
  case 52: // ASCII  value for '4'
    digitalWrite(IN_4, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_4, LOW);
    break;
  default:
    Serial.println("Not a valid case"); // prints out until a valid number has been
entered
    break;
  }
}
```

Try out different combinations of driving the four inputs until you find one that will drive the motor either clockwise or counter-clockwise. For example, if you find a sequence (A B C D), which will step the motor clockwise, then the reverse of the sequence (D C B A) will step the motor counter-clockwise, and vice versa. **(Lab-Report Q4) Write down the coil sequence in terms of the inputs into the H-bridge as described above. This sequence will come in handy later on in the lab.**

Eric's solution….
For example, to drive motor CCW, coil sequence = 1423, corresponding to IN_1, IN_4, IN_2, then IN_3 on the H-bridge. So, to drive motor CW, reverse the coil sequence = 3241.

`Serial.read()`. It can only return ASCII representations, meaning if user inputted a '1', it will return a '49'. So if the user input 200, it will only read the first digit ('2') and return a 50. So it will do 50 steps.

Exercise 2: Using the Arduino Stepper library

The Arduino software libraries come with a `Stepper` library, which will do the stepping sequence for you. Sample code for this library may be found at **http://arduino.cc/en/Tutorial/MotorKnob**. You will be using a potentiometer connected to Analog Pin 2 to control the position of the motor according to the value that is being read from the analog pin. An example of how to connect a potentiometer can be found at **http://www.arduino.cc/en/Tutorial/Potentiometer**.

Modify the code so that the correct pin is being read. You will also need to identify the pins that the motor connects to (notice that there are four numbers), the first two numbers represent the first coil of the stepper and the second two numbers represent the second coil of the stepper. If you did your setup properly, the pins in the code should correspond to your setup). Now modify the code so that the position of the stepper motor turns to the correct angle that the potentiometer is set to, for example, if your potentiometer turns 180 degrees, make sure that the stepper motor does the same. **(Lab-Report Q5) Include the code in your lab report.**

Eric's Solution….

(Lab-Report Q5) Now modify the code [the MotorKnob code from the Arduino.cc website https://www.arduino.cc/en/Tutorial/MotorKnob] so that the position of the stepper motor turns to the correct angle that the potentiometer is set to, for example, if your potentiometer turns 180 degrees, make sure that the stepper motor does the same.

1st, let's get the number of steps in 1 revolution

- 1.8 degrees comes from Appendix A (the Stepper Motor specs.) and it represents the number of degrees per 1 step.
- Declare the value as a constant: `#define STEPS 200`

$$\frac{360\ deg}{1\ revolution} \times \frac{1\ step}{1.8\ deg} = 200 \frac{steps}{rev} = 200\ steps\ per\ 1\ rev$$

```
/*
 *  This is Lab 8: Stepper Motor, Exercise 2, Question 5. This program is
 *  a modified version of the MotorKnob program of the Stepper libarary
 *  found on the Arduino.cc website https://www.arduino.cc/en/Tutorial/MotorKnob
 *
 *  The program was modified so that the position of the Stepper Motor turns
 *  to the correct angle that the potentiometer is set to. For example, if
 *  the potentiometer turns 180 degrees, the stepper motor turns to its 180
 *  degree mark
 */

#include <Stepper.h>
```

```
/* select the Analog input pin for the potentiometer */
#define POTENTIOMETER A1

/* change this to the number of steps on your motor
   Increasing this number increases the speed that the shaft turns.
   Specifically, this means more steps will be taken per 1 revolution.

   This was originally 100. But our Stepper Motor had 200 Steps per
   1 revolution because of the 1.8 degrees per step, found in
   Appendix A of the Manual.
   STEPS = (360 / 1.8 degrees) = 200 Steps per 1 revolution */
#define STEPS 200

/* create an instance of the stepper class, specifying
   the number of steps of the motor and the pins it's
   attached to

   The 1st two numbers represent the 1st coil of the Stepper.
   The 2nd two numbers represent the 2nd coil of the Stepper.
   Digital Pin 8 and Digital Pin 9 are connected to IN_1 and IN_2.
   IN_1 and IN_2 correlate to OUT_1 and OUT_2, which are connected
   to the red and blue wires of the Stepper */
Stepper stepper(STEPS, 8, 9, 10, 11);

/* the previous reading from the analog input */
int previous = 0;

void setup()
{
  Serial.begin(9600);
  /* set the speed of the motor to 30 RPMs */
  stepper.setSpeed(30);
}

void loop()
{
  /* get the potentiometer reading */
  int val = analogRead(POTENTIOMETER);
  Serial.println("\n Potentiometer Reading: " + String(val));
  /* Convert potentiometer reading (0 to 1023) into Degrees (0 to 360) to print out */
  Serial.println("Stepper angle: " + String(val/2.8416666667));
  /* Convert potentiometer reading (0 to 1023) into Steps (0 to 200) and turn the Stepper
motor accordingly */
  val = map(val, 0, 1023, 0, 200);
  Serial.println("Steps Taken (200 Steps = 1 rev): " + String(val));
  /* move a number of steps equal to the change in the sensor reading */
  stepper.step(val - previous);
  /* remember the previous value of the sensor */
  previous = val;
}
```

Exercise 3: Stepping the Motor/Wave Mode

The simplest way to drive a stepper motor is in *wave mode*, also called *one-phase-on drive.* This coil sequencing is shown in Figure 4 where the two colors represent the two different magnetic poles. The rotor of a real stepper motor has many more magnetized pole 'teeth' than that shown in the figure, which has ten poles.

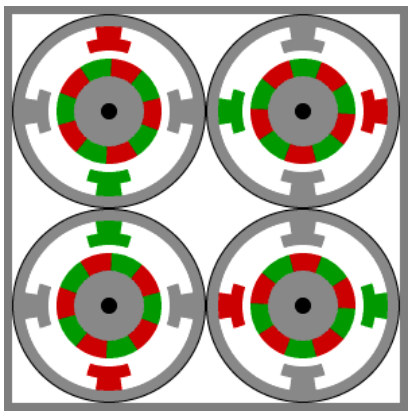

Figure 4. Full stepping of a rotor at its rated step angle. Note that only one phase is on at any one point in time. Step sequence for regular (one-phase-on) drive mode. This method steps the rotor at it its rated step resolution.

Use the data you gathered in the first exercise for the step sequence, and write a small program which will step through the sequence that you did manually in **Exercise 1**. Remember to turn on the enables for both A and B of the motor driver if the jumpers are not present. The delay between steps is critical in determining the rotational speed and torque of the motor. Since the rotor inertia prevents the motor from instantly moving to its new location, a stepper motor will not work if the energization of the stator phases are not properly timed between steps. In this exercise you will be experimentally finding out what the minimum delay (in microseconds) that is possible for the stepper motor to run under the no-load condition in lab. **(Lab-Report Q6) Record the value and include it in your lab report.**

Eric's Solution….

(Lab-Report Q6) Use the data you gathered in the first exercise for the step sequence and write a small program which will step through the sequence that you did manually in Exercise 1. The delay between steps is critical in determining the rotational speed and torque of the motor. Since the rotor inertia prevents the motor from instantly moving to its new location, a stepper motor will not work if the energization of the stator phases are not properly timed between steps. In this exercise you will be experimentally finding out what the minimum delay (in microseconds) that is possible for the stepper motor to run under the no-load condition in lab.

Find the minimum delay (the delay between energizing the coil and turning it off), so that the motor doesn't "twitch". A good starting point is 6000, but experiment with different values by changing `const int minDelay = 6000` to 7000, or 8000, etc. (Note: this will be in microseconds).

```
/*
 *  This is Lab 8: Stepper Motor, Exercise 3, Question 6.
 *  Using the coil sequence from Exercise 1, the program will:
 *
 *  Step through the sequence that you did manually in Exercise 1.
 */


// define the connections to motor driver inputs
#define IN_1 8
```

```
#define IN_2 9
#define IN_3 10
#define IN_4 11

int serialData = 0;

int CCW [4] = {IN_1, IN_4, IN_2, IN_3};
int CW [4] = {IN_3, IN_2, IN_4, IN_1};

/* Found the minimum delay (the delay between energizing the
   coil and turning it off), so that the motor doesn't "twitch" */
/* Delay of 6000 microseconds = 0.006 sec */
const int minDelay = 6000;

void setup()
{
  // begin serial communication
  Serial.begin(9600);
  // setup the pins to motor driver as outputs
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);
}

void loop()
{

  for(int i = 0; i < 4; i++)
  {
    digitalWrite(CCW[i], HIGH);
    delayMicroseconds(minDelay);
    digitalWrite(CCW[i], LOW);
  }
}
```

Now modify your code to make it a function (if you have not done so already) called `steps_wave()`, so that it takes an integer value as an input and moves the motor the appropriate number of steps. If you supply a negative number for the parameter, the motor must spin in the counterclockwise direction. This method of driving a single coil at a time is the *wave mode,* as described above. **(Lab-Report Q7)** **Include the code in your lab report.**

Note to self: Still trying to find a way to have the program take user-inputted number of steps from the Serial Monitor and move the motor the number of steps inputted. However, there is a problem with `Serial.read()`. It can only return ASCII representations, meaning if user inputted a '1', it will return a '49'. So if the user input 200, it will only read the first digit ('2') and return a 50. So it will do 50 steps.

For now, just manually pass in the number of steps by inputting it into the parameter of steps_wave() in the setup(), so it will only run once.

Also Note: Don't put a Serial.print after digitalWrite(IN, LOW);. It will make stall the program and make the motor twitch.

Eric's Solution….

```
/*
 *  This is Lab 8: Stepper Motor, Exercise 3, Question 7:
 *  Wave Drive Mode.
```

```
 *  Using the coil sequence from Exercise 1, the program will:
 *
 *  Contain a function called steps_wave(), so that it takes an
 *  integer value as an input and moves the motor the appropriate
 *  number of steps. If you supply a negative number for the
 *  parameter, the motor must spin in the counterclockwise direction.
 */


/* (I) Include libraries */
// stdlib.h is a C Standard General Utilities Library/Header
// http://www.cplusplus.com/reference/cstdlib/
// To use the atoi(serialData) function in the loop()
#include <stdlib.h>


/* (II) Variables */
// define the connections to motor driver inputs
#define IN_1 8
#define IN_2 9
#define IN_3 10
#define IN_4 11

int serialData = 0;

int CW [4] = {IN_3, IN_2, IN_4, IN_1};
int CCW [4] = {IN_1, IN_4, IN_2, IN_3};

/* Delay of 6000 microseconds = 0.006 sec */
const int minDelay = 6000;


/* (III) Function Prototypes */
void steps_wave(int mySteps);


/* (IV) setup */
void setup()
{
  // begin serial communication
  Serial.begin(9600);
  // setup the pins to motor driver as outputs
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);

  /* Call the steps_wave function once. Rotate 200 steps (1 revolution) CW. */
  steps_wave(200);
  /* Call the steps_wave function once. Rotate 200 steps (1 revolution) CCW. */
  steps_wave(-200);
}


/* (V) loop */
void loop()
{
  /*
  Serial.println("Input the number of steps to step the motor.");
  // look for when data is available for read from the serial monitor console
  if (Serial.available() > 0)
  {
    // read in the the serial data to a variable
    serialData = Serial.read();
    // Convert String (ASCII representation) to int
    //serialData = atoi(serialData);
```

```
    // also print out serial data entered
    Serial.println("You inputted: " + String(serialData));
    // Call the steps_wave function, and pass user-inputted Data as a parameter
    steps_wave(serialData);
  }
  */
}


/* (VI) Function Definitions */
void steps_wave(int stepsToMake)
{
  /* Current step number (used as a counter) */
  int mySteps = 0;

  /* If user inputted a pos number, turn the stepper CW */
  if(stepsToMake > 0)
  {
    while(mySteps <= stepsToMake)
    {
      for(int i = 0; i < 4; i++)
      {
        digitalWrite(CW[i], HIGH);
        delayMicroseconds(minDelay);
        digitalWrite(CW[i], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps++;
      }
    }
  }

  /* If user inputted a neg number, turn the stepper CCW */
  if(stepsToMake < 0)
  {
    while(mySteps >= stepsToMake)
    {
      for(int j = 0; j < 4; j++)
      {
        digitalWrite(CCW[j], HIGH);
        delayMicroseconds(minDelay);
        digitalWrite(CCW[j], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps--;
      }
    }
  }

}
```

Exercise 4: High-Torque Mode

There is a simple way to get more torque from the motor without requiring any additional hardware. This method of driving the motor is called high-torque mode and works on the principle of energizing two coils at the same time instead of one as shown in **Figure 5**. Doing so gives you an increase in torque of approximately 30%. Write a function, `steps_hitorque()`, that implements high-torque mode. Once again, a negative value indicates that the motor should be driven in the opposite direction. **(Lab-Report Q8) Include this code in your report.**

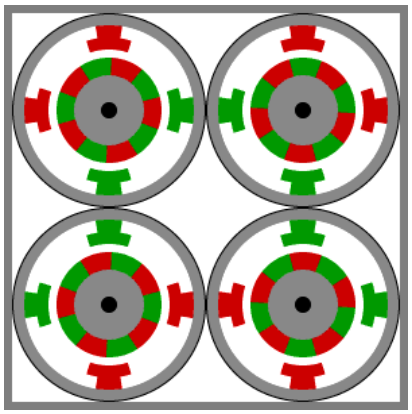

Figure 5. Demonstration of full-stepping in high-torque mode. Note that in all four states, both coils are driven simultaneously, generating more torque than is produced in wave mode, at the same step resolution.

Eric's Solution….

In this exercise, the code from Exercise #3 is modified to energize more than one coil at a time. Essentially, by having one coil pushing the rotor and the other coil pulling the rotor, more torque can be developed.

Note to self: Still trying to find a way to have the program take user-inputted number of steps from the Serial Monitor and move the motor the number of steps inputted. However, there is a problem with `Serial.read()`*. It can only return ASCII representations, meaning if user inputted a '1', it will return a '49'. So if the user input 200, it will only read the first digit ('2') and return a 50. So it will do 50 steps.*

For now, just manually pass in the number of steps by inputting it into the parameter of steps_wave() in the setup(), so it will only run once.

Also Note: Don't put a Serial.print after digitalWrite(IN, LOW);. It will make stall the program and make the motor twitch.

```
/*
 *  This is Lab 8: Stepper Motor, Exercise 4, Question 8:
 *  High-Torque Mode
 *  Using the coil sequence from Exercise 1, the program will:
 *
 *  Contain a function called steps_hitorque() that implements
 *  high-torque mode, which works on the principle of energizing
 *  two coils at the same time (Figure 5) instead of one. Once
 *  again, a negative value indicates that the motor should be
 *  driven in the opposite direction (CCW direction).
 */


/* (I) Include libraries */
// stdlib.h is a C Standard General Utilities Library/Header
// http://www.cplusplus.com/reference/cstdlib/
// To use the atoi(serialData) function in the loop()
#include <stdlib.h>


/* (II) Variables */
// define the connections to motor driver inputs
#define IN_1 8
#define IN_2 9
```

```
#define IN_3 10
#define IN_4 11

int serialData = 0;

int CW_coilA [4] = {IN_1, IN_2, IN_2, IN_1};
int CW_coilB [4] = {IN_3, IN_3, IN_4, IN_4};

int CCW_coilA [4] = {IN_1, IN_2, IN_2, IN_1};
int CCW_coilB [4] = {IN_4, IN_4, IN_3, IN_3};

/* Delay of 6000 microseconds = 0.006 sec */
const int minDelay = 6000;


/* (III) Function Prototypes */
void steps_hitorque(int mySteps);


/* (IV) setup */
void setup()
{
  // begin serial communication
  Serial.begin(9600);
  // setup the pins to motor driver as outputs
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);

  /* Call the steps_hitorque function once. Rotate 200 steps (1 revolution) CW. */
  steps_hitorque(200);
  /* Call the steps_hitorque function once. Rotate 200 steps (1 revolution) CCW. */
  steps_hitorque(-200);
}


/* (V) loop */
void loop()
{
  /*
  Serial.println("Input the number of steps to step the motor.");
  // look for when data is available for read from the serial monitor console
  if (Serial.available() > 0)
  {
    // read in the the serial data to a variable
    serialData = Serial.read();
    // Convert String (ASCII representation) to int
    //serialData = atoi(serialData);
    // also print out serial data entered
    Serial.println("You inputted: " + String(serialData));
    // Call the steps_wave function, and pass user-inputted Data as a parameter
    steps_wave(serialData);
  }
  */
}


/* (VI) Function Definitions */
void steps_hitorque(int stepsToMake)
{
  /* Current step number (used as a counter) */
  int mySteps = 0;

  /* If user inputted a pos number, turn the stepper CW */
  // int CW_coilA [4] = {IN_1, IN_2, IN_2, IN_1};
```

```
  // int CW_coilB [4] = {IN_3, IN_3, IN_4, IN_4};
  if(stepsToMake > 0)
  {
    while(mySteps <= stepsToMake)
    {
      for(int i = 0; i < 4; i++)
      {
        digitalWrite(CW_coilA[i], HIGH);
        digitalWrite(CW_coilB[i], HIGH);
        delayMicroseconds(minDelay);
        digitalWrite(CW_coilA[i], LOW);
        digitalWrite(CW_coilB[i], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps++;
      }
    }
  }

  /* If user inputted a neg number, turn the stepper CCW */
  // int CCW_coilA [4] = {IN_1, IN_2, IN_2, IN_1};
  // int CCW_coilB [4] = {IN_4, IN_4, IN_3, IN_3};
  if(stepsToMake < 0)
  {
    while(mySteps >= stepsToMake)
    {
      for(int j = 0; j < 4; j++)
      {
        digitalWrite(CCW_coilA[j], HIGH);
        digitalWrite(CCW_coilB[j], HIGH);
        delayMicroseconds(minDelay);
        digitalWrite(CCW_coilA[j], LOW);
        digitalWrite(CCW_coilB[j], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps--;
      }
    }
  }

}
```

Exercise 5: Half-stepping

Half-stepping is achieved by advancing the rotor through all of the positions present in both wave mode and high-torque mode. A pictorial representation is shown in **Figure 6**. With this type of sequencing, it should be apparent that the variety of positional outputs is twice the resolution of the rated step size. **(Lab-Report Q9) With respect to the output torque characteristics of wave mode and high-torque mode what, if any, downsides might be present when driving a stepper in this fashion?**

As seen in Figure 6, half-stepping a stepper motor entails alternating between energizing 2 coils, then 1 coil, then 2 coils, then 1, etc. In other words, we are alternating between high-torque mode (2 coils) and wave drive mode (1 coil). Therefore, one downside to half-stepping is that the torque of the motor would be uneven. The torque from the 2-phase energization (high-torque mode) is greater than the torque from the 1-phase energization (wave drive mode). As a result, the motor may become unstable, twitch or jerk, and produce lots of vibration, making it unsuitable for systems require minimal vibration.

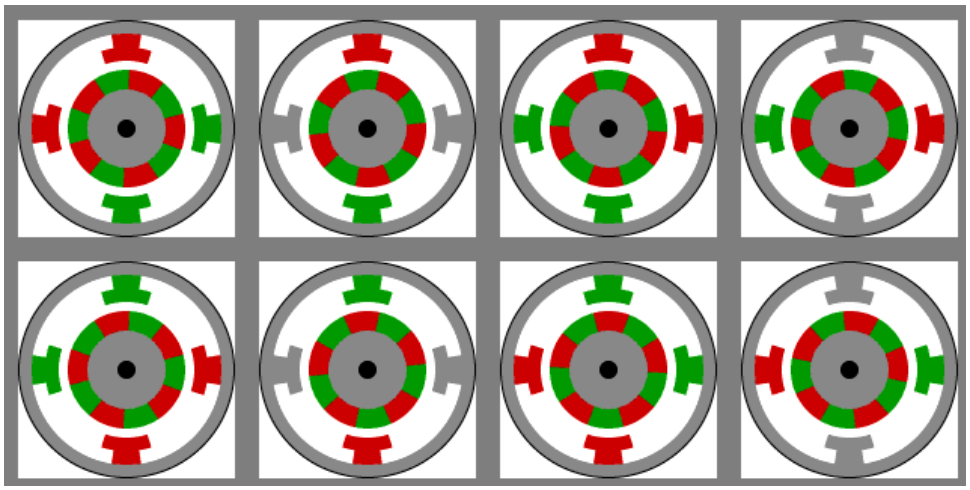

Figure 6. Demonstration of half-stepping. Note that we go through the positions from wave mode and high-torque mode.

In this exercise you will be using what you learned in Exercises 3 and 4 to write a function called `steps_half()` to increment the motor at double the resolution (half-stepping) that the motor is rated for, **Show your results to TA**. **(Lab-Report Q10) Include this code in you lab report.**

Eric's Solution….

The process of half stepping involves driving the motor in a hybrid one and two phase system where the motor alternates between two and one phase drive. This system enables us to double the number of steps that the motor is able to take, resulting in more fluid motion and resolution, but at the cost of increased control complexity. Also, with finer resolution comes less torque.

Note to self: Still trying to find a way to have the program take user-inputted number of steps from the Serial Monitor and move the motor the number of steps inputted. However, there is a problem with `Serial.read()`. It can only return ASCII representations, meaning if user inputted a '1', it will return a '49'. So if the user input 200, it will only read the first digit ('2') and return a 50. So it will do 50 steps.

For now, just manually pass in the number of steps by inputting it into the parameter of steps_wave() in the setup(), so it will only run once.

Also Note: Don't put a Serial.print after digitalWrite(IN, LOW);. It will make stall the program and make the motor twitch.

```
/*
 *  This is Lab 8: Stepper Motor, Exercise 5, Question 9:
 *  Half-Step Drive Mode
 *  Using the coil sequence from Exercise 1, the program will:
 *
 *  Contain a function called steps_half() to increment the
 *  motor at double the resolution (half-stepping) that the
 *  motor is rated for. This will involve what was learned in
 *  Exercises 3 and 4 (Wave drive mode and Hi-Torque mode).
 *  The motor will alternate netween 1 and 2 phase drive (Wave
 *  drive mode and Hi-Torque mode).
 *
 *  Half-stepping involves alternating between energizing 2 coils,
 *  then 1 coil, then 2 again, etc. This means alternating between
 *  high-torque mode (2 coils), then wave drive mode (1 coil), etc.
 */
```

```
/* (I) Include libraries */
// stdlib.h is a C Standard General Utilities Library/Header
// http://www.cplusplus.com/reference/cstdlib/
// To use the atoi(serialData) function in the loop()
#include <stdlib.h>


/* (II) Variables */
// define the connections to motor driver inputs
#define IN_1 8
#define IN_2 9
#define IN_3 10
#define IN_4 11

int serialData = 0;

/* off will be used as a placeholder in the Arrays to signify a coil to be OFF */
int off = 0;

/* CW Direction */
int CW_coilA [8] = {IN_1, IN_1, IN_1, off,  IN_2, IN_2, IN_2, off };
int CW_coilB [8] = {IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3, IN_3};
int power_coilA[8]={HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW };
int power_coilB[8]={HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH};

/* CCW Direction */
int CCW_coilA [8] = {off,  IN_2, IN_2, IN_2, off,  IN_1, IN_1, IN_1};
int CCW_coilB [8] = {IN_3, IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3};
int rev_coilA[8]  = {LOW , HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH};
int rev_coilB[8]  = {HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH};

/* Delay of 6000 microseconds = 0.006 sec */
const int minDelay = 6000;


/* (III) Function Prototypes */
void steps_half(int mySteps);


/* (IV) setup */
void setup()
{
  // begin serial communication
  Serial.begin(9600);
  // setup the pins to motor driver as outputs
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);

  /* Call the steps_half function once. Rotate 400 microsteps (1 revolution) CW.
     Must double the steps because it is half-stepping  */
  steps_half(400);
  /* Call the steps_half function once. Rotate 400 microsteps (1 revolution) CCW.
     Must double the steps because it is half-stepping. */
  steps_half(-400);
}


/* (V) loop */
void loop()
{
  /*
  Serial.println("Input the number of steps to step the motor.");
  // look for when data is available for read from the serial monitor console
```

```
  if (Serial.available() > 0)
  {
    // read in the the serial data to a variable
    serialData = Serial.read();
    // Convert String (ASCII representation) to int
    //serialData = atoi(serialData);
    // also print out serial data entered
    Serial.println("You inputted: " + String(serialData));
    // Call the steps_wave function, and pass user-inputted Data as a parameter
    steps_wave(serialData);
  }
  */
}


/* (VI) Function Definitions */
void steps_half(int stepsToMake)
{
  /* Current step number (used as a counter) */
  int mySteps = 0;

  /* If user inputted a pos number, turn the stepper CW */
  // int CW_coilA [8] = {IN_1, IN_1, IN_1, off,  IN_2, IN_2, IN_2, off };
  // int CW_coilB [8] = {IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3, IN_3};
  // int power_coilA[8]={HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW };
  // int power_coilB[8]={HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH};
  if(stepsToMake > 0)
  {
    while(mySteps <= stepsToMake)
    {
      for(int i = 0; i < 8; i++)
      {
        digitalWrite(CW_coilA[i], power_coilA[i]);
        digitalWrite(CW_coilB[i], power_coilB[i]);
        delayMicroseconds(minDelay);
        digitalWrite(CW_coilA[i], LOW);
        digitalWrite(CW_coilB[i], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps++;
      }
    }
  }

  /* If user inputted a neg number, turn the stepper CCW */
  // int CCW_coilA [8] = {off,  IN_2, IN_2, IN_2, off,  IN_1, IN_1, IN_1};
  // int CCW_coilB [8] = {IN_3, IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3};
  // int rev_coilA[8]  = {LOW , HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH};
  // int rev_coilB[8]  = {HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH};
  if(stepsToMake < 0)
  {
    while(mySteps >= stepsToMake)
    {
      for(int j = 0; j < 8; j++)
      {
        digitalWrite(CCW_coilA[j], rev_coilA[j]);
        digitalWrite(CCW_coilB[j], rev_coilB[j]);
        delayMicroseconds(minDelay);
        digitalWrite(CCW_coilA[j], LOW);
        digitalWrite(CCW_coilB[j], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps--;
      }
    }
  }

}
```

(Extra Credit:) Exercise 6 (Optional): Micro-stepping

Start by removing the jumpers from the EN_A and EN_B pins on the L298 H-bridge motor driver board. Then connect EN_A and EN_B to either Digital Pins 6, 5, or 3 since Digital Pins 11, 10, 9, 6, 5, and 3 are the only pins that can perform Pulse-Width-Modulation (PWM), but Digital Puns 11 and 10 are currently occupied by the IN_3 and IN_4 pins.

Micro-stepping entails:
Analog-writing the EN_A and EN_B pins at a specific duty cycle

We are going to:
```
analogWrite(EN_A, either 0% DC 50% DC or 100% DC)
analogWrite(EN_B, either 0% DC 50% DC or 100% DC)
```

```
/*
 *  This is Lab 8: Stepper Motor, Exercise 6, EXTRA CREDIT:
 *  Micro-stepping Mode
 *  Using the coil sequence from Exercise 1, the program will:
 *
 *  Contain a function called steps_micro() to step the motor
 *  in micro-steps
 */


/* (I) Include libraries */
// stdlib.h is a C Standard General Utilities Library/Header
// http://www.cplusplus.com/reference/cstdlib/
// To use the atoi(serialData) function in the loop()
#include <stdlib.h>


/* (II) Variables */
// Define the connections to ENABLE pins on motor driver
#define EN_A 5
#define EN_B 6
// define the connections to motor driver inputs
#define IN_1 8
#define IN_2 9
#define IN_3 10
#define IN_4 11

int serialData = 0;

/* off will be used as a placeholder in the Arrays to signify a coil to be OFF */
int off = 0;

/* CW Direction */
int CW_coilA [10] = {IN_1, IN_1, IN_1, off,  IN_2, IN_2, IN_2, off,  IN_1, IN_1};
int CW_coilB [10] = {IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3, IN_3, IN_3, off };
int power_coilA[10]={HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH};
int power_coilB[10]={HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW };
int dutyCycle_ENA[10]={127,255,  127,  0,    127,  255,  127,  0   , 127,  255 };
int dutyCycle_ENB[10]={127,  0,  127,  255,  127,    0,  127,  255 , 127,  0   };

/* CCW Direction */
int CCW_coilA [10] = {off,  IN_2, IN_2, IN_2, off,  IN_1, IN_1, IN_1, off,  IN_2};
int CCW_coilB [10] = {IN_3, IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3, IN_3, IN_3};
int rev_coilA[10]  = {LOW , HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH};
int rev_coilB[10]  = {HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH};
int backCycle_ENA[10]={  0, 255,  127,  255,    0,  255,  127,  255,  0,    255 };
int backCycle_ENB[10]={127, 255,    0,  255,  127,  255,    0,  255,  127,  255 };
```

```
/* Delay of 6000 microseconds = 0.006 sec */
const int minDelay = 6000;


/* (III) Function Prototypes */
void steps_micro(int mySteps);


/* (IV) setup */
void setup()
{
  // begin serial communication
  Serial.begin(9600);
  // setup the ENABLE pins on motor driver as outputs
  pinMode(EN_A, OUTPUT);
  pinMode(EN_B, OUTPUT);
  // setup the pins to motor driver as outputs
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);

  /* Call the steps_half function once. Rotate 400 microsteps (1 revolution) CW.
     Must double the steps because it is half-stepping  */
  steps_micro(400);
  /* Call the steps_half function once. Rotate 400 microsteps (1 revolution) CCW.
     Must double the steps because it is half-stepping. */
  steps_micro(-400);
}


/* (V) loop */
void loop()
{
  /*
  Serial.println("Input the number of steps to step the motor.");
  // look for when data is available for read from the serial monitor console
  if (Serial.available() > 0)
  {
    // read in the the serial data to a variable
    serialData = Serial.read();
    // Convert String (ASCII representation) to int
    //serialData = atoi(serialData);
    // also print out serial data entered
    Serial.println("You inputted: " + String(serialData));
    // Call the steps_wave function, and pass user-inputted Data as a parameter
    steps_wave(serialData);
  }
  */
}


/* (VI) Function Definitions */
void steps_micro(int stepsToMake)
{
  /* Current step number (used as a counter) */
  int mySteps = 0;

  /* If user inputted a pos number, turn the stepper CW */
  // int CW_coilA [8] = {IN_1, IN_1, IN_1, off,  IN_2, IN_2, IN_2, off };
  // int CW_coilB [8] = {IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3, IN_3};
  // int power_coilA[8]={HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW };
  // int power_coilB[8]={HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH, HIGH};
  // int dutyCycle_ENA[8]={127,255,  127,  0,    127,  255,  127,  0   };
  // int dutyCycle_ENB[8]={127,  0,  127,  255,  127,    0,  127,  255 };
```

```
  if(stepsToMake > 0)
  {
    while(mySteps <= stepsToMake)
    {
      for(int i = 0; i < 8; i++)
      {
        analogWrite(EN_A, dutyCycle_ENA[i]);
        analogWrite(EN_B, dutyCycle_ENB[i]);
        digitalWrite(CW_coilA[i], power_coilA[i]);
        digitalWrite(CW_coilB[i], power_coilB[i]);
        delayMicroseconds(minDelay);
        digitalWrite(CW_coilA[i], LOW);
        digitalWrite(CW_coilB[i], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps++;
      }
    }
  }

  /* If user inputted a neg number, turn the stepper CCW */
  // int CCW_coilA [8] = {off,  IN_2, IN_2, IN_2, off,  IN_1, IN_1, IN_1};
  // int CCW_coilB [8] = {IN_3, IN_3, off,  IN_4, IN_4, IN_4, off,  IN_3};
  // int rev_coilA[8]  = {LOW , HIGH, HIGH, HIGH, LOW,  HIGH, HIGH, HIGH};
  // int rev_coilB[8]  = {HIGH, HIGH, LOW,  HIGH, HIGH, HIGH, LOW,  HIGH};
  // int backCycle_ENA[8]={  0, 255,  127,  255,    0,  255,  127,  255 };
  // int backCycle_ENB[8]={127, 255,    0,  255,  127,  255,    0,  255 };
  if(stepsToMake < 0)
  {
    while(mySteps >= stepsToMake)
    {
      for(int j = 0; j < 8; j++)
      {
        analogWrite(EN_A, backCycle_ENA[j]);
        analogWrite(EN_B, backCycle_ENB[j]);
        digitalWrite(CCW_coilA[j], rev_coilA[j]);
        digitalWrite(CCW_coilB[j], rev_coilB[j]);
        delayMicroseconds(minDelay);
        digitalWrite(CCW_coilA[j], LOW);
        digitalWrite(CCW_coilB[j], LOW);
        // Serial.println("Step count is currently at: " + String(mySteps));
        mySteps--;
      }
    }
  }

}
```

You may have noticed that all combinations of inputs are represented when performing half-stepping with fully-on and fully-off currents in each coil. There are no other combinations of inputs that can set the rotor in a position other than the eight just presented. However, it should be apparent that if one is willing to drive coils with fractional currents, that in theory any position of the rotor can be achieved. As has been demonstrated numerous times throughout the semester, one can produce other than fully-on and fully-off outputs by using PWM to produce fractional outputs between the two digital extremes.

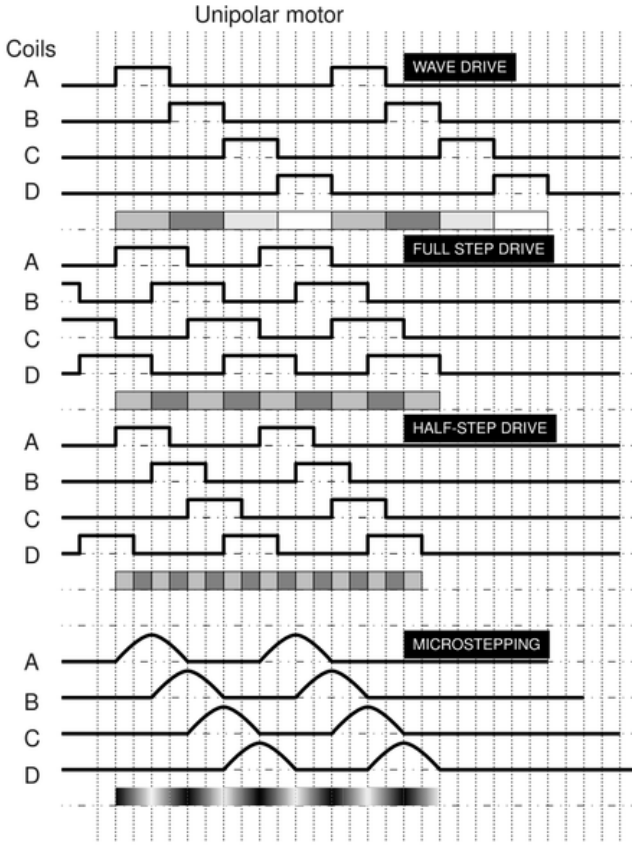

Figure 7: Graphical comparison of drive modes for a unipolar stepper motor (Source: http://upload.wikimedia.org/wikipedia/commons/8/85/Drive.png)

Micro-stepping is an important part in many electromechanical applications of stepper motors which require very small resolution from the stepper motor without adding any additional mechanical hardware such as gear reduction. Micro-stepping involves pulsing the enable pin (if using an H-bridge in a bipolar configuration, or the current-control transistor's gate for unipolar) at a specified duty cycle to generate an intermediate voltage. By doing so, you can get the rotor to stop between its natural half-step positions. As shown in Figure 7; the difference in power in each of the coils can potentially generate infinite resolution from a stepper motor if the coils are driven with sinusoidal current waveforms.

Start by creating a table that details the fractional drive level in each coil for each step in the (16-step) sequence for a full rotation. See **Appendix D** for an idea of what this would look like. Once completed, implement quarter-step micro-stepping using the Arduino, **show your results to TA** and **(Lab-Report Extra)** **include the code in your lab report**.

References

Stepping Motors Fundamentals: http://ww1.microchip.com/downloads/en/AppNotes/00907a.pdf

Stepper Motor Guide: http://www.anaheimautomation.com/manuals/forms/stepper-motor-guide.php

Appendix A

Motor No.: 5718L-10D

WINDNG REVISION	**ECN#** 0
FULL STEP ANGLE	1.8°
MOTOR FRAME SIZE	NEMA Size 23
BODY LENGTH	3.1" Maximum
RATED CURRENT/PHASE	1Amps
RESISTANCE/PHASE	8.2Ω ±10%
DIELECTRIC STRENGTH	500V
ROTOR INERTIA	2.6oz-in^2
BEARING TYPE	ABEC3
NOMINAL HOLDING TORQUE 2 PHASE ON (±10%)	193 oz-in
AMBIENT OPERATING TEMPERATURE	-20° C to 50° C
AMBIENT OPERATING HUMIDITY(non condensing)	85% Max
LEAD WIRES	(6) # 22 AWG PVC

LEAD WIRE CONNECTION

A	RED
A/C	WHITE
C	BLUE
B	GREEN
B/D	YELLOW
D	BLACK

Switching Sequence					
CW	STEP	A	C	B	D
↓	1	ON	OFF	ON	OFF
	2	OFF	ON	ON	OFF
	3	OFF	ON	OFF	ON
	4	ON	OFF	OFF	ON
	1	ON	OFF	ON	OFF
Motor Rotation Viewed from Front Shaft End					

Appendix B

6 LEAD WIRES	1	2	3	4	5	6
Color Code 1	Red	White	Blue	Green	Yellow	Black
Color Code 2	Brown	Black	Orange	Red	White	Yellow
Color Code 3	Red	Black	Red / White Stripe	Green	White	Green / White Stripe
Bipolar Drive Half Coil Connection	A	$\overline{A}$		B	$\overline{B}$	
		$\overline{A}$	A		$\overline{B}$	B
Bipolar Drive Series Connection	A		$\overline{A}$	B		$\overline{B}$
Unipolar Drive	A	A/C Comm	C	B	B/D Comm	D

Appendix C

```
// define the connections to motor driver inputs
#define IN_1 8
#define IN_2 9
#define IN_3 10
#define IN_4 11
// define the delay time for steps
#define DELAY_TIME 30

int serialData = 0;

void setup()
{
  // begin serial communication
  Serial.begin(9600);
  // setup the pins to motor driver as outputs
  pinMode(IN_1, OUTPUT);
  pinMode(IN_2, OUTPUT);
  pinMode(IN_3, OUTPUT);
  pinMode(IN_4, OUTPUT);
}

void loop()
{
  // look for when data is available for read from the serial monitor console
  if (Serial.available() > 0)
  {
    serialData = Serial.read();  // read in the the serial data to a variable
    Serial.println(serialData);  // also print out serial data entered
  }

  // use variable as a switch in which the ASCII characters 1 through 4 correponds to
each of the pins
  switch (serialData)
  {
  case 49: // ASCII  value for '1'
    digitalWrite(IN_1, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_1, LOW);
    break;
  case 50: // ASCII  value for '2'
    digitalWrite(IN_2, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_2, LOW);
    break;
  case 51: // ASCII  value for '3'
    digitalWrite(IN_3, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_3, LOW);
    break;
  case 52: // ASCII  value for '4'
    digitalWrite(IN_4, HIGH);
    delay(DELAY_TIME);
    digitalWrite(IN_4, LOW);
    break;
  default:
    Serial.println("Not a valid case"); // prints out until a valid number has been
entered
    break;
  }
}
```

Appendix D

A3955 *Full-Bridge PWM Microstepping Motor Driver*

Table 4 — Step Sequencing

Full Step	Half Step	Quarter Step	Eighth Step	Bridge A					Bridge B				
				$PHASE_A$	D_{2A}	D_{1A}	D_{0A}	I_{LOADA}	$PHASE_B$	D_{2B}	D_{1B}	D_{0B}	I_{LOADB}
1	1	1	1	H	H	L	L	70.7%	H	H	L	L	70.7%
			2	H	L	H	H	55.5%	H	H	L	H	83.1%
		2	3	H	L	H	L	38.2%	H	H	H	L	92.4%
			4	H	L	L	H	19.5%	H	H	H	H	100%
	2	3	5	X	L	L	L	0%	H	H	H	H	100%
			6	L	L	L	H	-19.5%	H	H	H	H	100%
		4	7	L	L	H	L	-38.2%	H	H	H	L	92.4%
			8	L	L	H	H	-55.5%	H	H	L	H	83.1%
2	3	5	9	L	H	L	L	-70.7%	H	H	L	L	70.7%
			10	L	H	L	H	-83.1%	H	L	H	H	55.5%
		6	11	L	H	H	L	-92.4%	H	L	H	L	38.2%
			12	L	H	H	H	-100%	H	L	L	H	19.5%
	4	7	13	L	H	H	H	-100%	X	L	L	L	0%
			14	L	H	H	H	-100%	L	L	L	H	-19.5%
		8	15	L	H	H	L	-92.4%	L	L	H	L	-38.2%
			16	L	H	L	H	-83.1%	L	L	H	H	-55.5%
3	5	9	17	L	H	L	L	-70.7%	L	H	L	L	-70.7%
			18	L	L	H	H	-55.5%	L	H	L	H	-83.1%
		10	19	L	L	H	L	-38.2%	L	H	H	L	-92.4%
			20	L	L	L	H	-19.5%	L	H	H	H	-100%
	6	11	21	X	L	L	L	0%	L	H	H	H	-100%
			22	H	L	L	H	19.5%	L	H	H	H	-100%
		12	23	H	L	H	L	38.2%	L	H	H	L	-92.4%
			24	H	L	H	H	55.5%	L	H	L	H	-83.1%
4	7	13	25	H	H	L	L	70.7%	L	H	L	L	-70.7%
			26	H	H	L	H	83.1%	L	L	H	H	-55.5%
		14	27	H	H	H	L	92.4%	L	L	H	L	-38.2%
			28	H	H	H	H	100%	L	L	L	H	-19.5%
	8	15	29	H	H	H	H	100%	X	L	L	L	0%
			30	H	H	H	H	100%	H	L	L	H	19.5%
		16	31	H	H	H	L	92.4%	H	L	H	L	38.2%
			32	H	H	L	H	83.1%	H	L	H	H	55.5%