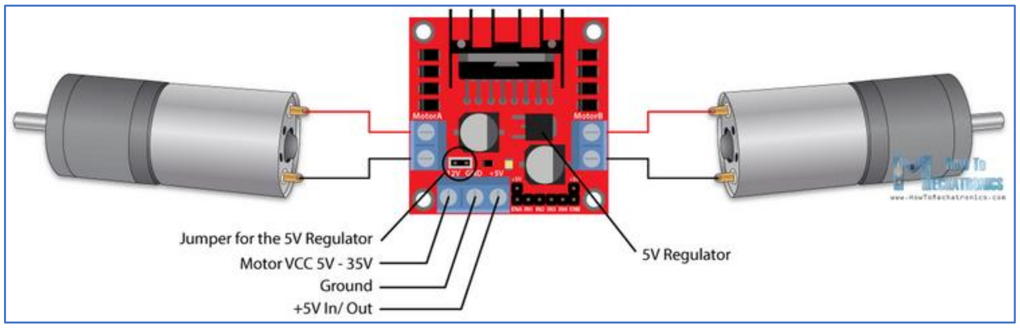
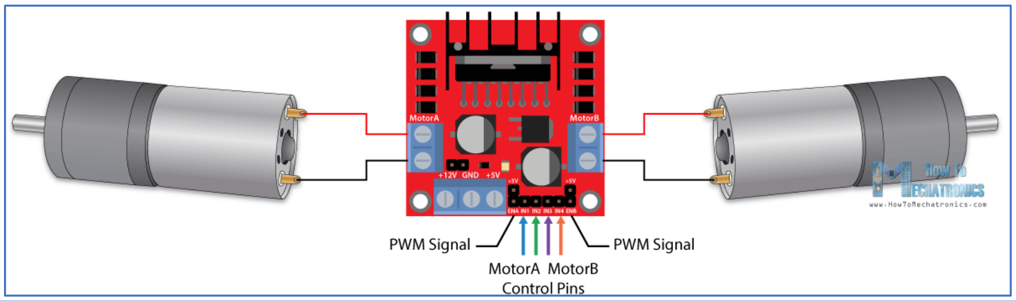
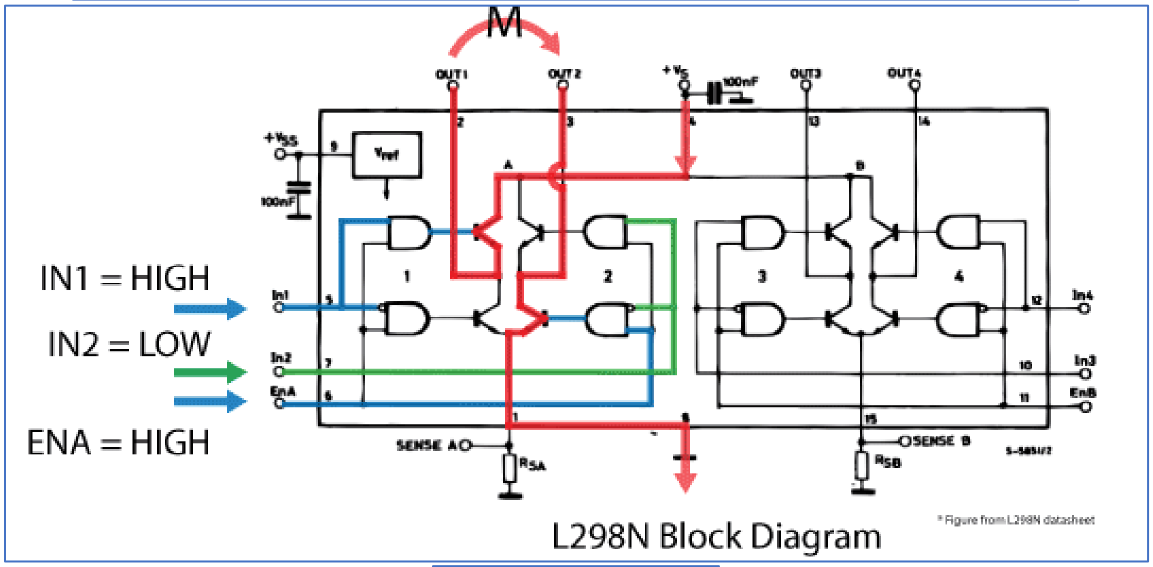
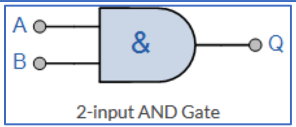
Lab 8 Websites:

L298 Motor Driver Board:
https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/









Learning Objectives:

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

- Describe the advantages of stepper motors
    - Although they cannot produce continuous rotations, Stepper motors operate at low voltage
    - However, they can achieve quasi-continuous rotation through microstepping (the best refinement of steps achievable)
    - Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms (Arduino.cc).
    - https://www.applied-motion.com/videos/encoder-feedback-step-motor-systems
        - Can accelerate at precise rates
        - Can stop in an instant with no overshoot
        - Stepper motors are low-cost
    - http://www.anaheimautomation.com/manuals/forms/stepper-motor-guide.php#sthash.pPL8t3xu.dpbs
        - Cost-effective*
            - *Stepper motor products vary in cost based on the criteria for each application. Some criteria include options of 0.9°, 1.8°, 3.6° and 4.5° step angles, torque ranging from 1 to 5,700 oz-in, and NEMA frame sizes of 08 to 42. Additional attachments such as cables and encoders can be purchased separately for an additional cost. With our friendly customer service and professional application assistance, Anaheim Automation often surpasses customer expectations for fulfilling specific stepper motor and driver requirements, as well as other motion control needs.
        - Simple designs
        - High reliability
        - Brushless construction
        - Maintenance-free
        - If windings are energized at standstill, the motor has full torque
        - No feedback mechanisms required
        - High acceleration and power rate
        - A wide range of rotational speeds can be attained as the speed is proportional to the frequency of the input pulses
        - Known limit to the dynamic position error
- Describe the disadvantages of stepper motors
    - The more refined the steps are, the less torque is available
    - http://www.anaheimautomation.com/manuals/forms/stepper-motor-guide.php#sthash.pPL8t3xu.dpbs
        - Low efficiency (Motor attracts a substantial amount of power regardless of the load)
        - Torque drops rapidly with speed (torque is inversely proportional of speed)
        - Prone to resonance* (Microstepping allows for smooth motion)
            - *Resonance-is inherent in the design and operation of all stepping motors and occurs at specific step rates. It is the combination of slow stepping rates, high rotor inertia, and elevated torque which produce ringing as the rotor overshoots its desired angular displacement and is pulled back into position causing resonance to occur. Adjusting either one of the three parameters –inertial load, step rate, or torque- will reduce or eliminate resonance. In practical practice, the torque parameter is more controllable using microstepping. In microstepping mode, power is applied to the stator windings incrementally which causes torque to slowly build, reducing overshoot and therefore reducing resonance.
        - No feedback to indicate missed steps
        - Low torque-to-inertia ratio
        - Cannot accelerate loads very rapidly
        - Motor gets very hot in high performance configurations
        - Motor will not "pick up" after momentary overload
        - Motor is noisy at moderate to high speeds
        - Low output power for size and weight
- 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

Stepper Motor Wiring Configurations

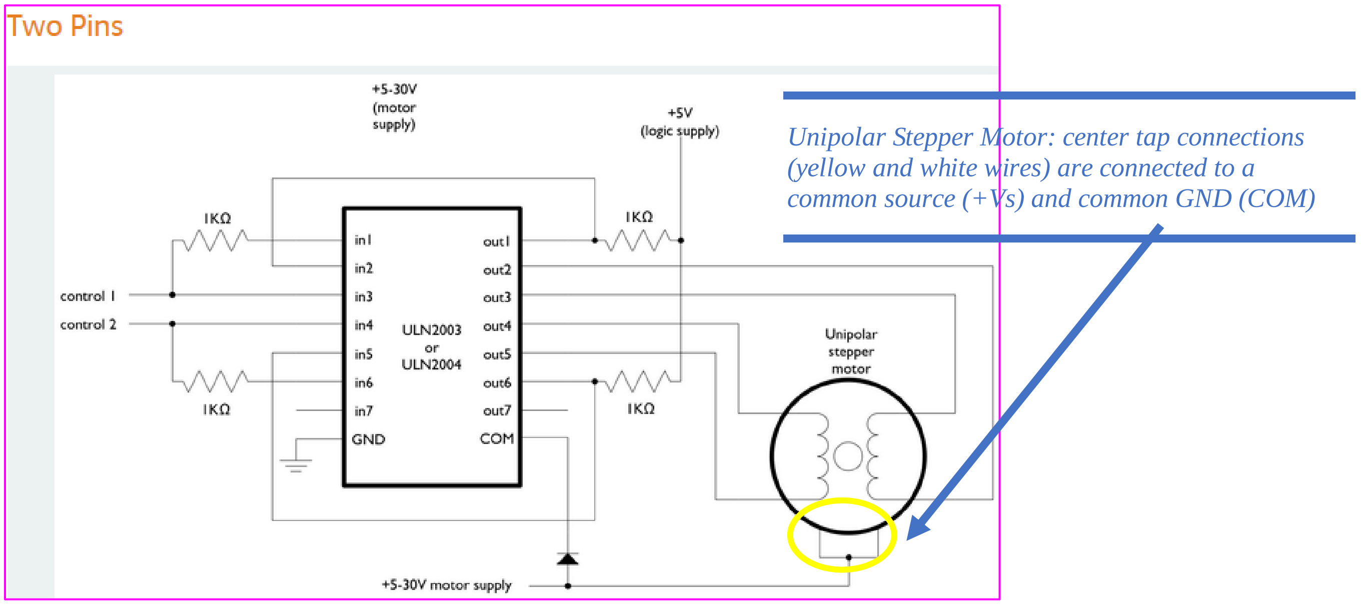
Unipolar stepper motor = connect coils in Parallel [Uni- means "one"]
Bipolar stepper motor = connect coils in Series [Bi- means "both", "double", or "twice"]
Universal stepper motor = unipolar-bipolar hybrid. Contains 4 independent coils (each coil has 2 wires) and thus 8 leads total.

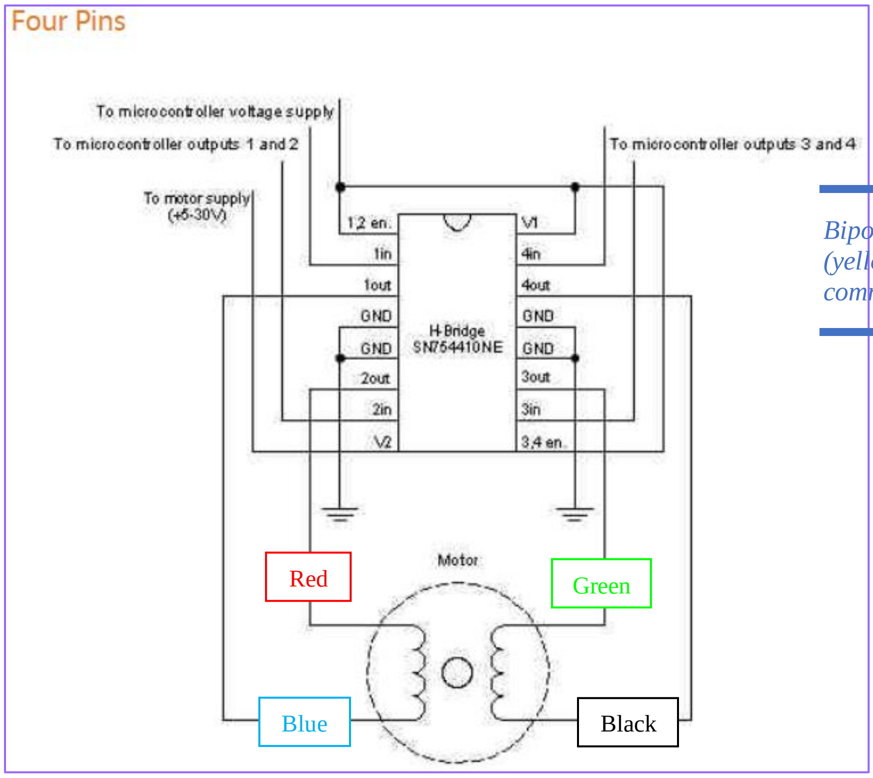
(Lab-Report Q1) A unipolar stepper motor can be configured as a bipolar stepper motor by disconnecting the center tap connections, i.e. do NOT connect the center tap connections to a common source. Therefore, the first set of coils (red and blue wires of the stepper) will be connected to each other, 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.

Images below from:
https://www.arduino.cc/en/Reference/StepperUnipolarCircuit
https://www.arduino.cc/en/Reference/StepperBipolarCircuit

(Lab-Report Q2)

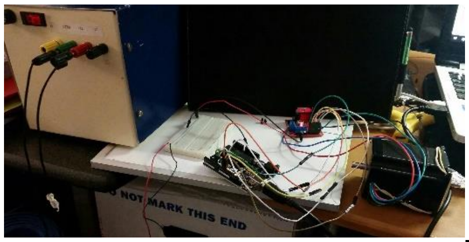
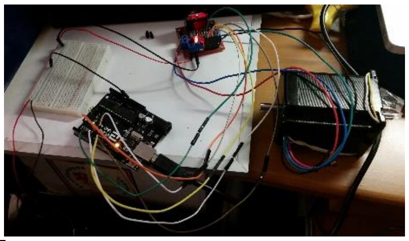
Number of Wires	Type of Stepper Motor
4	Bipolar
6	Unipolar
8	Universal





Bipolar Stepper Motor: center tap connections (yellow and white wires) are NOT connected to a common source and common GND

Exercise 1: Setup and Test

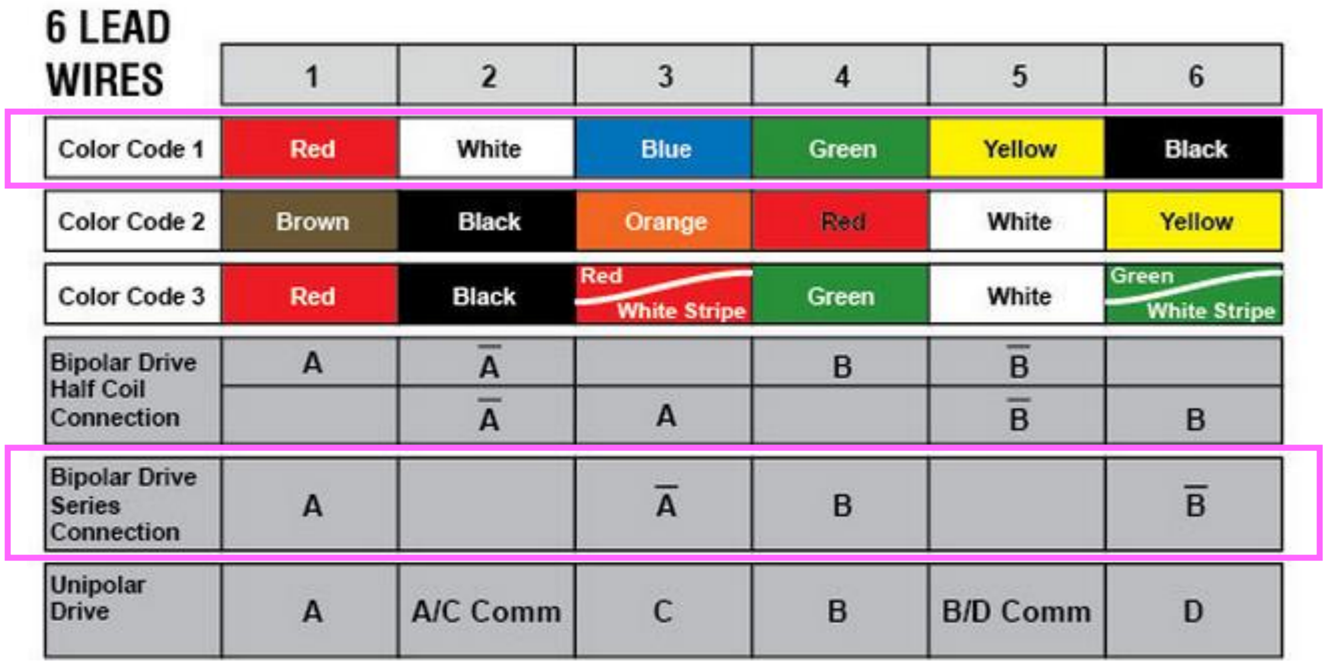
3.) If jumpers are present on the 5V regulator, and the enable (EN_A and EN_B) headers, can simply follow setup exactly. If jumpers are not present, must follow setup seen in Lab 7:

- Create a +5 V rail on the breadboard
    - Connect the +5 V power terminal on the L298 driver board to the +5 V rail
    - Connect the +5 V pin on the Arduino to the +5 V rail. The Arduino will be providing the +5 V that the L298 needs.
- Connect EN_A to a Digital Pin on the Arduino (used Digital Pin 10 in Lab 7)
- Connect EN_B to a Digital Pin on the Arduino
- In your code, you must digitalWrite(EN_A, HIGH) and digitalWrite(EN_B, HIGH); to be able to activate the IN_1 through IN_4 pins.

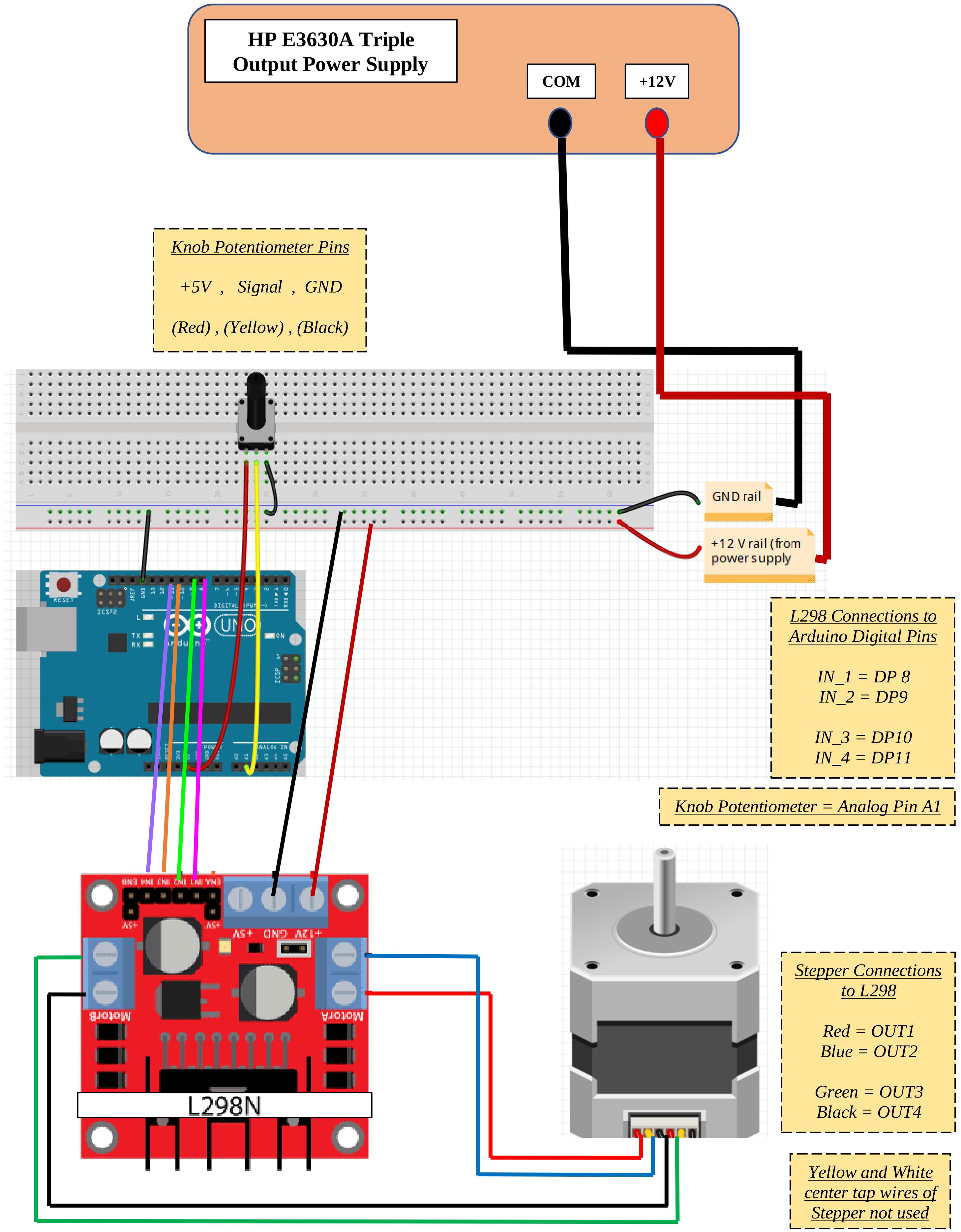
Note: you can simply directly connect the +12 V terminal on the L298 to the +12 V cable of the power supply, and the GND from the L298 and from the Arduino to the power supply. However, a 12 V rail and a GND rail will have to be made in the next exercise.

4.) We will be connecting this stepper motor in a bipolar series configuration. The white and yellow center tap wires from the stepper motor will not be connected to anything.

From Appendix B (Stepper Motor Wires):



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	$\bar{A}$		B	$\bar{B}$	
		$\bar{A}$	A		$\bar{B}$	B
Bipolar Drive Series Connection	A		$\bar{A}$	B		$\bar{B}$
Unipolar Drive	A	A/C Comm	C	B	B/D Comm	D


HP E3630A Triple Output Power Supply
COM
+12V
Knob Potentiometer Pins
+5V , Signal , GND
(Red) , (Yellow) , (Black)
GND rail
+12 V rail (from power supply
L298 Connections to Arduino Digital Pins
IN_1 = DP 8
IN_2 = DP9
IN_3 = DP10
IN_4 = DP11
Knob Potentiometer = Analog Pin A1
UNO
L298N
Stepper Connections to L298
Red = OUT1
Blue = OUT2
Green = OUT3
Black = OUT4
Yellow and White center tap wires of Stepper not used

(Lab-Report Q3)

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;
  }
}
```

(Lab-Report Q4)

Upload code from Appendix C to Arduino. Turn on power supply (+12 V will be supplied to the L298 H-bridge motor driver board). Run code. Open Serial Monitor. Start typing in different combinations of 1, 2, 3, and 4 to determine coil sequence that will move the motor CW or CCW. Each stepper motor will have a different coil sequence. You'll know if the motor is moving CCW or CW if the motor does not "twitch"; the tape on the shaft is there to show if it's moving CCW or CW.

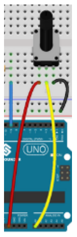
For example, to drive motor CCW, coil sequence = 1423, corresponding to IN_1, IN_4, IN_2, then IN_3 on the H-bridge. So, to drive motor CW, reverse the coil sequence = 3241.

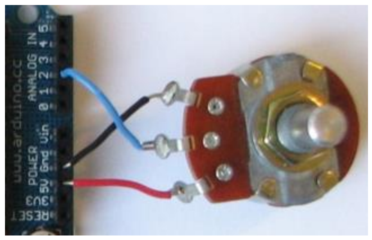
`Serial.read()`. It can only return ASCII representations, meaning if user inputted a '1', it will return a '49'. So if the user input 200, it will only read the first digit ('2') and return a 50. So it will do 50 steps.

Exercise 2: Using the Arduino Stepper library

For the potentiometer:

- Don't use a trim pot because they have finer resistance readings, meaning it will take forever to get one step, let alone one revolution
- Instead, Use the dial potentiometer on the EduShields shield, but if you do, make sure to take out the jumpers from Digital Pins 8, 9, 10, and 11, so that SW1, LED1/LEDGreen, SERVO, and LED0/LEDRed won't be connected, preventing possible damage.
- Or, Use a knob potentiometer:





(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.

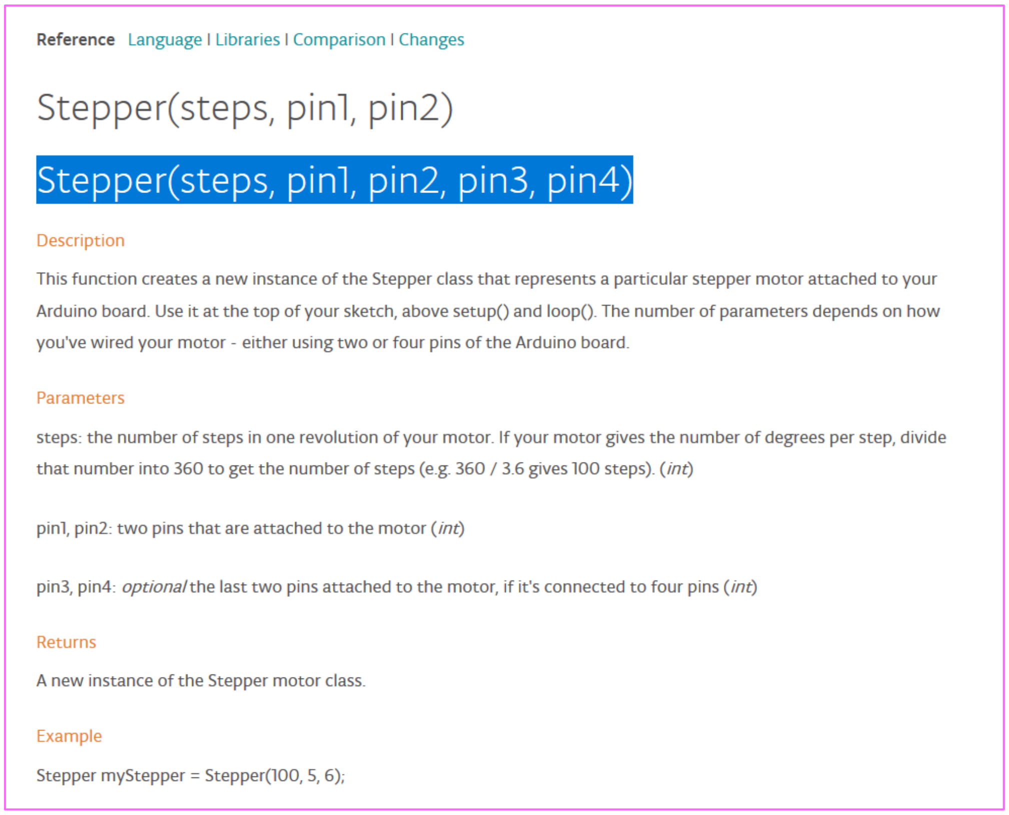
1st, let's get the number of steps in 1 revolution

- 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;
}
```

2nd, let's understand the Object Declaration: `Stepper stepper(STEPS, 8, 9, 10, 11);`

- 

Reference Language | Libraries | Comparison | Changes

Stepper(steps, pin1, pin2)

Stepper(steps, pin1, pin2, pin3, pin4)

Description

This function creates a new instance of the Stepper class that represents a particular stepper motor attached to your Arduino board. Use it at the top of your sketch, above setup() and loop(). The number of parameters depends on how you've wired your motor - either using two or four pins of the Arduino board.

Parameters

steps: the number of steps in one revolution of your motor. If your motor gives the number of degrees per step, divide that number into 360 to get the number of steps (e.g. 360 / 3.6 gives 100 steps). (*int*)

pin1, pin2: two pins that are attached to the motor (*int*)

pin3, pin4: *optional* the last two pins attached to the motor, if it's connected to four pins (*int*)

Returns

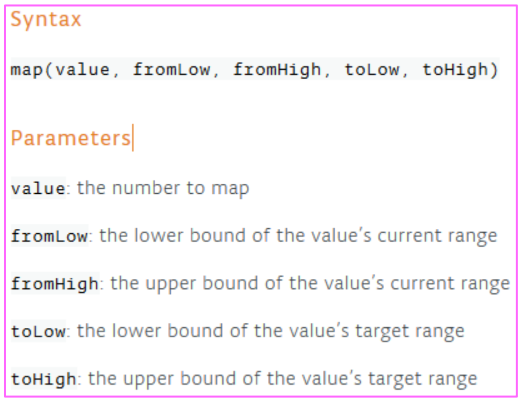
A new instance of the Stepper motor class.

Example

Stepper myStepper = Stepper(100, 5, 6);

3rd, let's recall the function: `val = map(val, 0, 1023, 0, 200);`

- Map basically creates a ratio (i.e. a proportion) between the fromLow and fromHigh to the → toLow and toHigh values. toLow and toHigh are the adjusted values.



Syntax

`map(value, fromLow, fromHigh, toLow, toHigh)`

Parameters

`value`: the number to map

`fromLow`: the lower bound of the value's current range

`fromHigh`: the upper bound of the value's current range

`toLow`: the lower bound of the value's target range

`toHigh`: the upper bound of the value's target range

4rd, let's understand the function: stepper.step()

- Turns the motor a specific number of steps, at a speed determined by the most recent call to setSpeed(). This function is blocking; that is, it will wait until the motor has finished moving to pass control to the next line in your sketch.
- Parameters
  - steps: the number of steps to turn the motor - positive to turn one direction, negative to turn the other (int)

Exercise 3: Stepping the Motor/Wave Mode

(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);
  }
}
```

(Lab-Report Q7) 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.

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 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

(Lab-Report Q8) Write a program that 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).

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

(Lab-Report Q9) 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. **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.

(Lab-Report Q10) **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.**

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--;
      }
    }
  }

}
```

Exercise 6 (EC): 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--;
      }
    }
  }

}
```