Stepper Motor Lab

# Learning Objectives:

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

* Describe the advantages and disadvantages of stepper motors versus 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

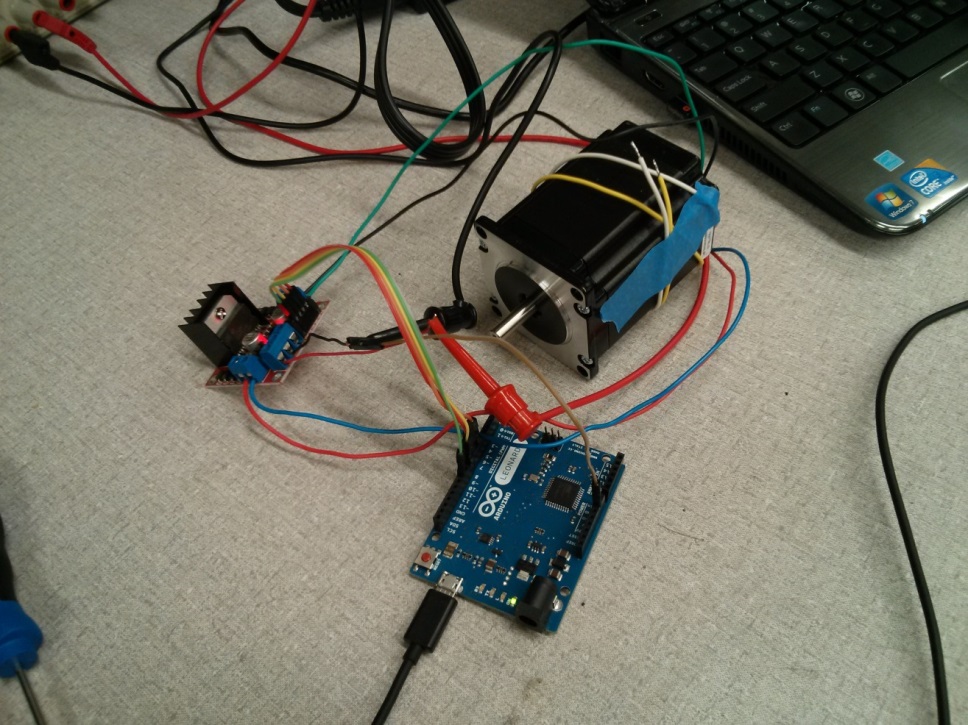
1 six-wire stepper motor with laser-cut position indicator wheel

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 ‘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 the motor driver board.



**Figure 1.** The stepper motor driver 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.

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.

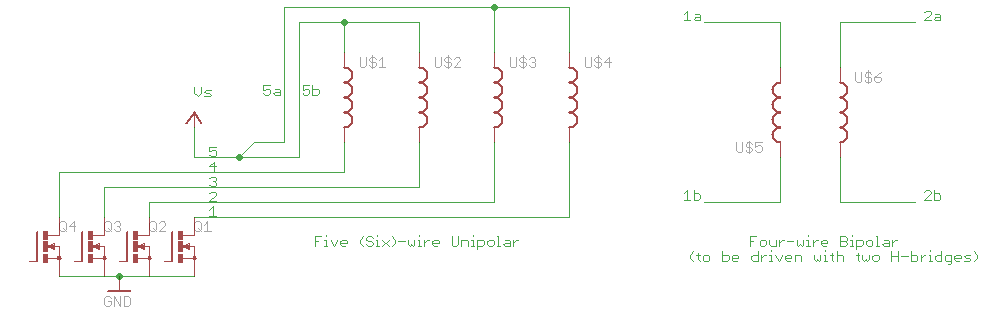


Figure 2. Unipolar vs. Bipolar Stepper Wiring Configuration

An outcome of these different wiring types is that 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. **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.

**In your lab report, create a table which one could use to identify a stepper motor based on the number of wires.**

One of the advantages of unipolar motors is that because only one end can be manipulated, a single transistor can be used to cause current to flow that single direction. In contrast, with a bipolar motor, current needs to be made to flow in both directions, an H-bridge is required to drive it. The advantage afforded to a bipolar stepper is that they have a better torque-to-size ratio, so their more complicated drive requirements provide an advantage 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.

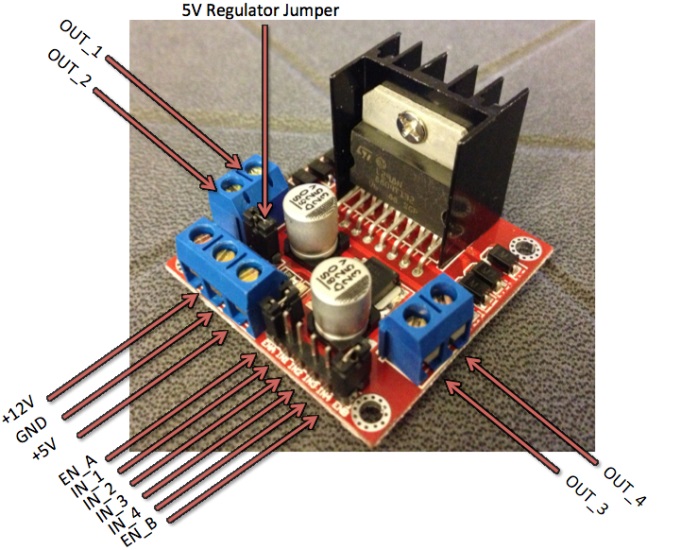
The other important parameter is the maximum current that should be used in the motor's coil. Fairly often this parameter is accompanied by a DC voltage rating for the motor, which is often specified to be 12V or 24V. By using Ohm's law, one can determine the DC resistance of the copper coil winding 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. For the purposes of this lab, 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.

# Exercise 1: Setup and Test



**Figure 3.** L298N Motor Driver Board (left) and fully connected driver board (right)

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 these are located). Skip to step 4 if your jumpers are present.
3. If the jumpers are not present, ask 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**).
   1. The white and yellow center tap wires from the stepper motor will not be connected to anything.
   2. Connect the red and blue wires from the stepper to OUT\_1 and OUT\_2 respectively.
   3. Connect the green and black wires from the stepper to OUT\_3 and OUT\_4 respectively.
   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.
   5. Now we need our power connections, set your +20V power supply to 12V, then complete the following sub-steps **with the Arduino and power supply powered off**.
      1. Run a wire from the +12V terminal of the motor driver board to the +20V power supply terminal
      2. Run another wire from the GND terminal of the motor driver board to the COM terminal of the power supply.
      3. Connect the Arduino’s GND to motor driver board’s GND terminal either by running a wire into the terminal or jumping them together on a breadboard.
   6. **Have the TA approve your setup before proceeding to power up the power supply and Arduino.**

Now we will want 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. (Look through the code and comments for further explanation on how this code works and what it does).

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. **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.**

# 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.

Remember to 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 4 numbers), the first 2 numbers represent the first coil of the stepper and the second 2 numbers represent the second coil second. 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. Include the code in your lab report.

# 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 the two colors represent the two different magnetic poles. The rotor of a real stepper motor has many more magnetized ‘teeth’ than that shown in the figure, which has ten poles.

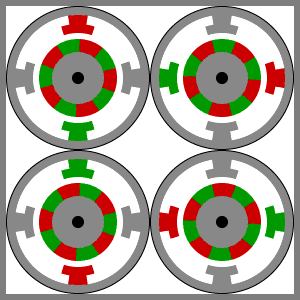


Figure 4. Full stepping of a rotor at its rated step angle. Note that only one phase is on at any one point in time. Step sequence for regular (one-phase-on) drive mode. This method steps the rotor at it its rated step resolution.

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. Record the value and include it in your lab report.

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. Include the code in your lab report.

# 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. Include this code in your report.

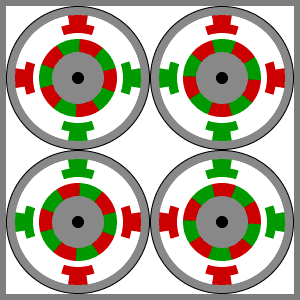


Figure 5. Demonstration of full-stepping in high-torque mode. Note that in all four states, both coils are driven simultaneously, generating more torque than is produced in wave mode, at the same step resolution.

# Exercise 6: 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. **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?**

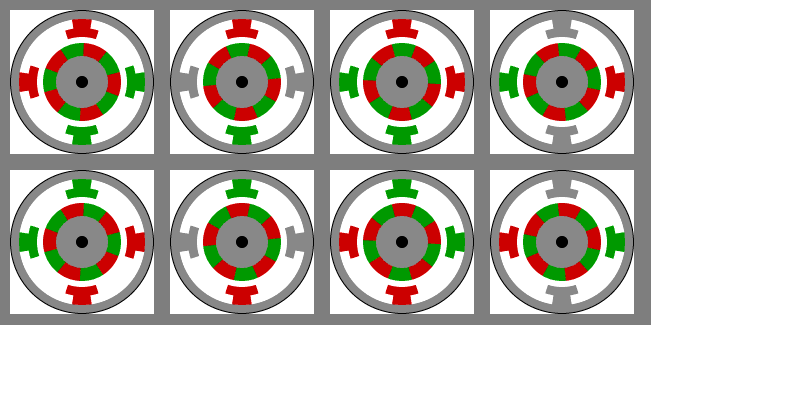
****

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 4 and 5 to write a function called steps\_half() to increment the motor at double the resolution (half-stepping) that the motor is rated for. Include this code in you lab report.

# Exercise 7 (Optional): Micro-stepping

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 were 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 (fake analog-equivalent) outputs between the two digital extremes.

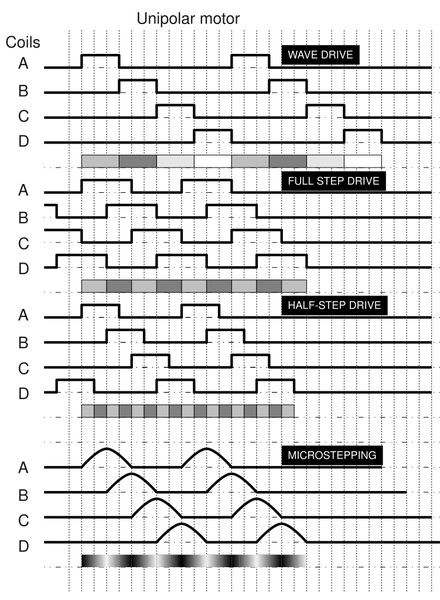


Figure 7: Graphical comparison of drive modes for a unipolar stepper motor (Source: http://upload.wikimedia.org/wikipedia/commons/8/85/Drive.png)

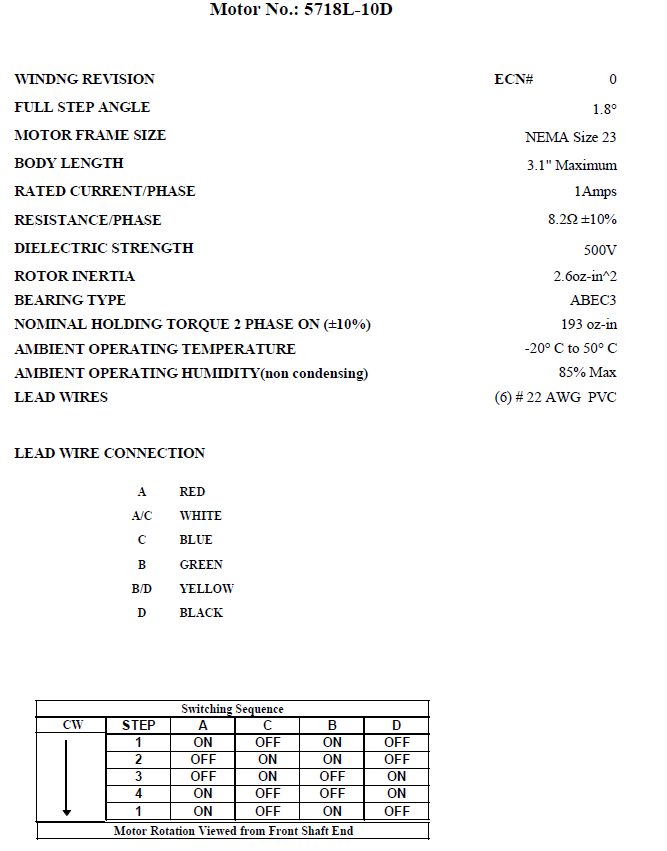
Micro-stepping is an important part in many electromechanical applications of stepper motors which require very small resolution from the stepper motor without adding any additional mechanical hardware such as gear reduction. Micro-stepping involves pulsing the enable pin (if using an H-bridge in a bipolar 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 and include the code in your lab report.

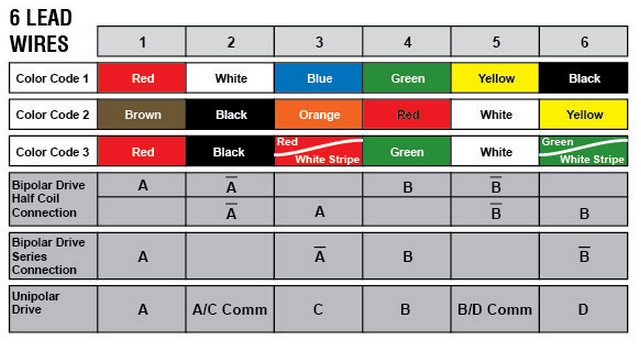
References

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**

**Appendix B**

****

**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;

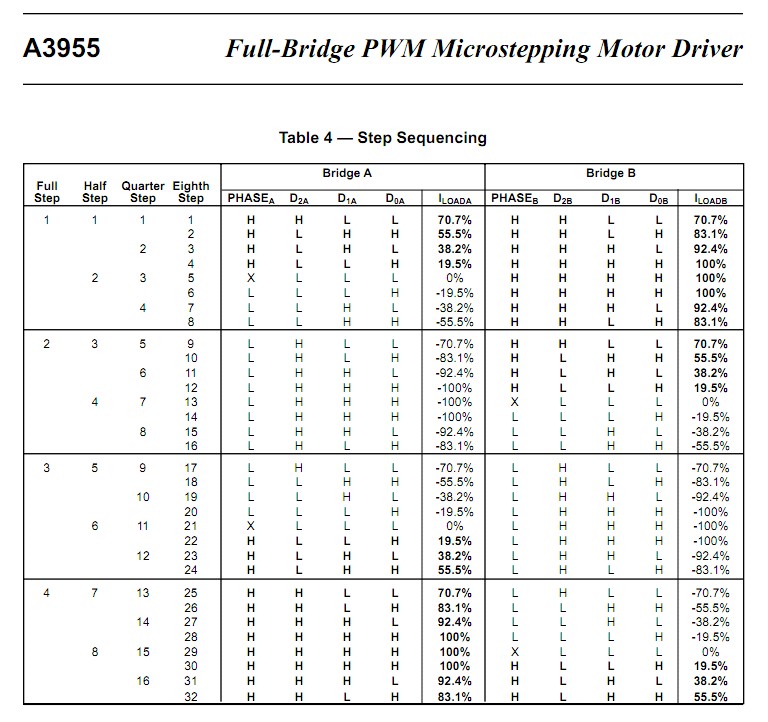
default:

Serial.println("Not a valid case"); // prints out until a valid number has been entered

break;

}

}

**Appendix D**