Skill Demo 6: Pulse Width Modulation and Motors

v1.4

Goals:

- Learn how Pulse Width Modulation (PWM) is produced
- Use PWM to dim and LED
- Use PWM to vary the speed of a DC brushed motor
- Drive a stepper motor

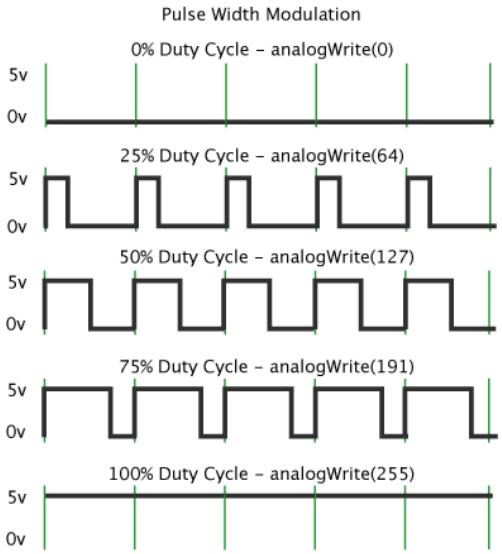
Tools/supplies:

- Various resistors
- Breadboard
- Wires
- 1x Arduino-compatible microcontroller development board
- 1x Speaker
- 1x DC brushed motor, ROB-10171 or similar
- 1x 600mA NPN BJT transistors, PN2222
- 1x Quad Half-H Bridge
- 1x unipolar stepper motor, PF35T-48L4 or 28BYJ-48

Background:

PWM

PWM, or Pulse Width Modulation, is a method of changing the 'duty cycle' or high-time of a rectangular wave to convey a value or affect an output circuit. You can think of a square wave as a special case of a PWM signal where the duty cycle is set at 50%.



PWM signals can be used for multiple things. At the simplest, you can use an RC filter to take the 'average' of a PWM signal, giving you an analog voltage that matches the duty cycle.

Additionally, PWM can be used to 'dim' LEDs. Since LED's can be turned on and off very quickly, a PWM signal with a frequency above 60Hz or so will appear to be dimmed. The LED will actually be flashing on and off, but too quickly for the human eye to see. (this is actually why many LED car tail lights appear to 'strobe' on slow motion video)

More commonly, PWM is used to control very simple ‘smart’ electronics. RC servos, often used as small, inexpensive actuators, as well as speed-controlled fans as can be found in most modern PCs, are both controlled by PWM. Various standards or de-facto standards are used for the frequency and duty cycle, depending on the application.

Because PWM is so commonly used, there is special hardware inside of many microcontrollers to make it easier to implement using less processor time. These are typically referred to as timer/counter modules, and you can read the details about them in chapter 19 of the ATmega 328p datasheet. This hardware is used by Arduino’s analogWrite() function, as well as the PWM and Servo libraries, and is the same hardware we used to for the timer interrupts in the ADC skill demo.

For a basic introduction on PWM within the Arduino framework, take a look at this page, or you can also look PWM tutorials like this one. For more on how PWM works ‘under the hood’ in most Arduinos, see here.

For more about control of RC servos with PWM, see Section 13.4 in the book, or this excellent primer from SparkFun.

Procedure:

For the following programs **make sure the PWM output pin has a tilda (~) next to it or is Identified by a PWM label on the PCB or it won't work!**

Attach a potentiometer to an analog input

For the rest of the skill demo, we'll be controlling LEDs, buzzers, and motors. For input, hook up your potentiometer to an analog input, just like in the ADC skill demo. Read it with AnalogRead() and output the value read to the serial console, just to make sure that's working.

1. Manually control the LED brightness with PWM

Connect an LED and current limiting resistor to a digital pin.

Using a loop, digitalWrite(), and delaymicroseconds(), manually generate a PWM signal (do not use analogWrite yet). Use a period of 4096 microseconds and a duty cycle of 50%. Toggle an LED on and off with this PWM signal.

Next, vary the PWM duty cycle based on the position of the potentiometer with analogRead(). Change the duty cycle from 0% to 100%.

Your code will probably follow this structure:

In a loop:
    potentiometer position = read analog input
    write the pin high
    delay microseconds(potentiometer position)
    write the pin low
    delay microseconds(max potentiometer position - potentiometer position)

Turn the potentiometer and show how the LED brightness changes.

(complete sign-off on the last page)

2. Control a Motor's Speed using analogWrite()

Just as PWM is used to control the brightness of the LED by adjusting the average power flowing through it, PWM can control the speed of a DC motor. Use the analogWrite() function to generate the PWM signal to drive the base of a transistor (with a 330 ohm series resistor to limit the base current), and use that to control a motor. This circuit will look very similar to one from the transistor skill demo.

You can also choose to use your H-bridge driver ICs, instead of using a discrete transistor. In this case you can use the PWM output signal to drive the EN input on a pair of pins of the H-bridge IC. That will cause the H-bridge to alternate between driving and 'coasting' the motor.

You may want to start by turning the motor on and off using a digitalWrite() first, as a test of your circuit.

Make a program that controls the motor speed using the position of the potentiometer. Make your program so that the first ~10% of the potentiometer's range leaves the motor entirely off, and the 10%-100% range varies the speed of the motor. This helps prevent putting a small amount of power through the motor when the potentiometer is turned close to zero.

You might want to put a small piece of tape, a wheel, or something else onto the shaft of your motor to make the speed easier to see.

(complete sign-off on the last page)

3. Generate tones with a PWM output using tone()

Hook up a speaker. Use Tone() to play five notes, pause, then play them again in a loop. (If you are a fan of Close Encounters of the Third Kind, I recommend G, A F, F (1 octave lower), C.)

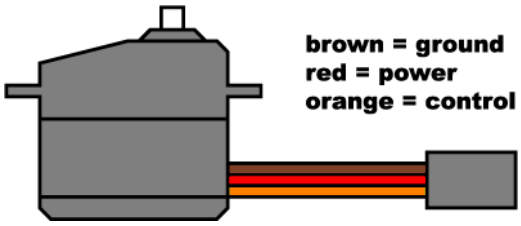
Use the small piezo speaker for audio output. Just connect it to the PWM output pin and ground.

The squarewave tones you hear from the PWM module are reminiscent of the 8-bit sound effects that accompanied video games of the 1970's and 1980's, as those were also driven by simple oscillators and square waves.

See the documentation for tone(). For note pitches to convert notes to frequency, read this link or find that information online.

(complete sign-off on the last page)

4. Control a servo with PWM, manually

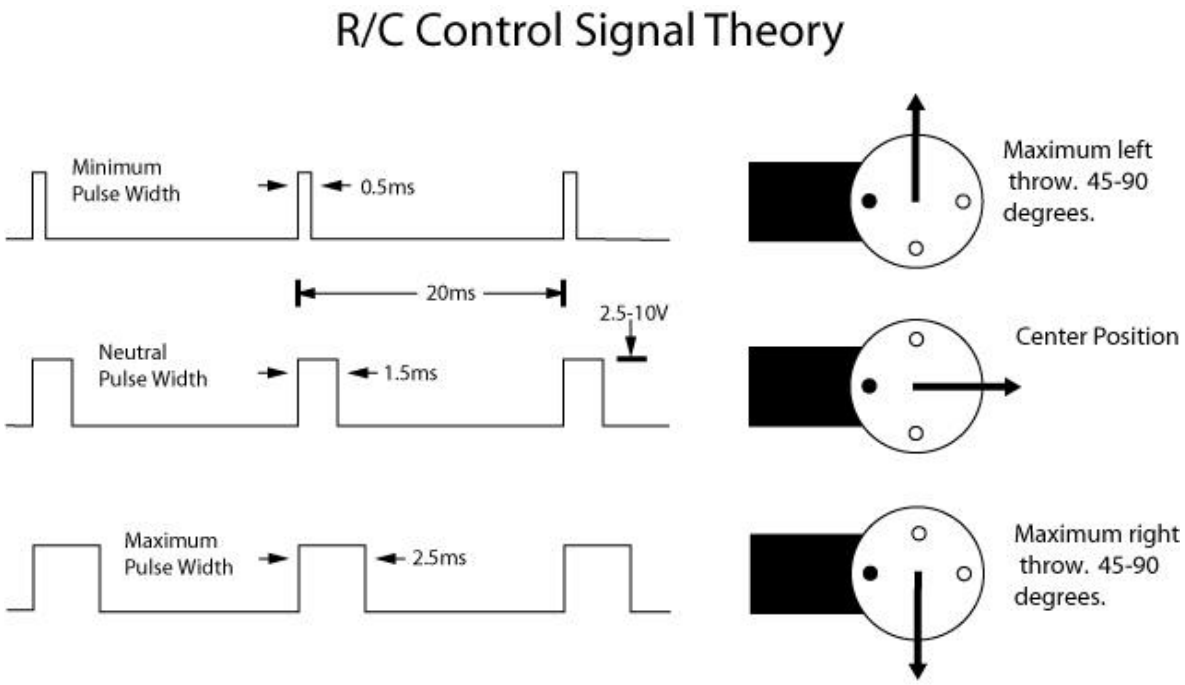


Small servo motors are commonly used in remote control vehicles to move parts such as steering wheels or ailerons. The position of the servo is controlled by sending a special PWM signal. The signal typically consists of a 1-2 ms high, then 19-18ms low. The total period of the signal should be 20ms.

The servo motor has three wires that connect to 5V, Ground, and Signal. Hook up the 5V and ground lines. Connect the control wire to a digital pin on the Arduino. You can use three jumper wires to connect the servo to your breadboard.

Using a while loop, digitalWrite(), and delaymicroseconds(), manually generate a PWM signal to move the servo. Use the value read from the potentiometer to control the angle of the servo arm. You may want to also leave an LED connected to this pin for debugging.

Note that you **may not** use a servo library in this part of the skill demo.
Everything you need to know about controlling the servo is in the following figure.



The minimum and maximum pulse width for different manufacturers can vary considerably; however, the neutral position is generally quite near 1.5ms regardless of manufacturer. Typical variance for the minimum pulse width is from 0.5ms to 0.8ms, and the typical variance for the maximum pulse width is from 2.5ms to 3.0ms. The frequency of the signal is generally near 50Hz; however, it can range from 30Hz to 200Hz. The output voltage can vary from 2.5V to as much as 10V.

For your reference, calculate the pulse time:

What is the total period of one cycle?

What is the minimum pulse width for maximum left servo position (top graph above)?

What is the maximum pulse width for maximum right servo position (bottom graph above)?

Position = angle in value from 0..1 (0 is max left, 1 is max right)

Pulse Width = Max left pulse width + (Max right pulse width - Max left pulse width) * Position

Pseudocode:

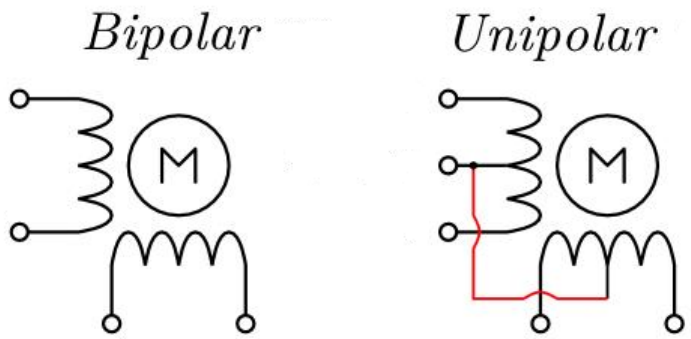
Set pin high
Pulse Width (in microseconds, use delayMicroseconds())
Set pin low
Cycle Period - Pulse Width (in microseconds, use delayMicroseconds())

Do not print anything to the serial port in this program, as this will add jitter to the signal timing, and the servo will buzz and jerk.

(complete sign-off on the last page)

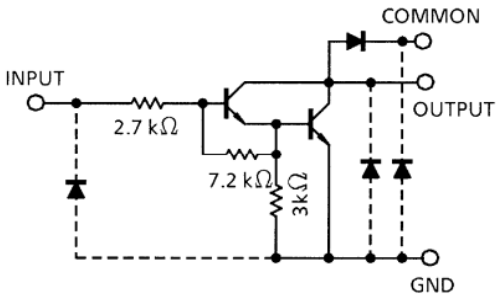
5. Stepper Motors

Next, set up a circuit to drive your stepper motor. Because these are unipolar stepper motors, you'll need to connect one or two 'common' wires to a voltage source, and then allow current to flow into the common wire and out of each of the winding wires in sequence.



Biploar and Unipolar stepper motors. Red wire may or may not exist.

Our kits contain small driver boards with a ULN2003APG darlington driver on it. This IC has 7 darlington pair drivers that will each sink 500 mA. This can't be used as an H-bridge, but is great for driving the coils of a unipolar stepper motor. This driver board also has the right 5-pin connector to easily plug the stepper motor into, so you can connect up Power, Ground, and 4 GPIO pins to use it.



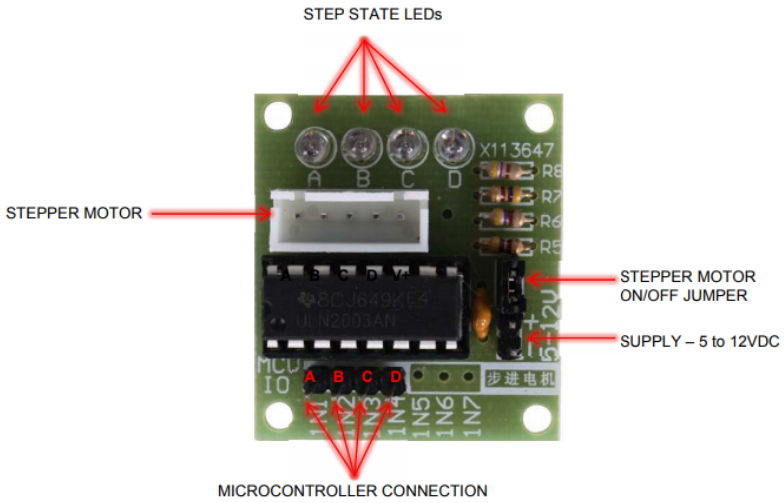
There are alternative ways to drive the stepper motor. One way to do this is with discrete transistors, such as BJT's or MOSFETs. Another way is to use the four half-H bridges in your H-bridge IC's. Depending on your stepper motor, you may be able to drive it as a bipolar stepper motor as well.

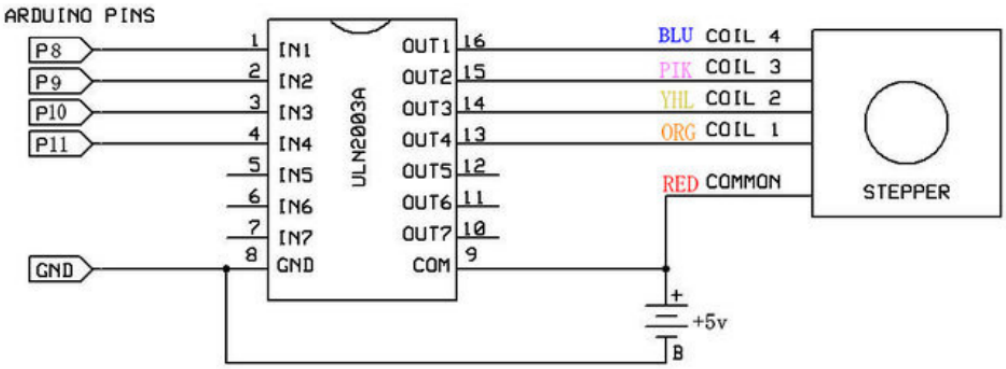
To test it, write a program to:

- Drive your stepper motor forward at a rate of 60 steps per second for 2 seconds
- Wait for 1 second
- Drive it backwards at 30 steps per second for 4 seconds
- Wait 1 second
- Repeat

To drive your stepper motor forwards and backwards, you'll need to drive the coils in the correct sequence at a controlled rate. You may need to take some measurements with a multimeter or experiment with trial and error to find the right sequence.

(complete sign-off on the last page)


STEP STATE LEDs
STEPPER MOTOR
STEPPER MOTOR
ON/OFF JUMPER
SUPPLY – 5 to 12VDC
MICROCONTROLLER CONNECTION


ARDUINO PINS
P8
P9
P10
P11
GND
IN1
IN2
IN3
IN4
IN5
IN6
IN7
GND
ULN2003A
OUT1
OUT2
OUT3
OUT4
OUT5
OUT6
OUT7
COM
BLU COIL 4
PIK COIL 3
YHL COIL 2
ORG COIL 1
RED COMMON
STEPPER
+5v

SD6: Pulse Width Modulation and Motors Turn-in Page

V1.4

NAME_____ GTID_____

☐ CS3651A
☐ CS3651B

1. Manually control the LED brightness with PWM

20/100 Initials_____ Date_____

2. Control a Motor's Speed using analogWrite()

20/100 Initials_____ Date_____

3. Generate tones with a PWM output using tone()

20/100 Initials_____ Date_____

4. Control a servo with PWM, manually

20/100 Initials_____ Date_____

5. Drive a stepper motor

20/100 Initials_____ Date_____