Build-It-Yourself Shops Document – Module 4

Intended Use & Scope

This document is developed by Technovus as an aid for members attending the workshops. The scope and context of the topics covered here is limited to embedded programming (C/C++), especially programming Arduino microcontrollers. The Build-It-Yourself Shops consist of six modules. Each module ends with a building exercise which will provide a way to practically use the information provided in the module.

Module 1 is designed to quickly introduce programming to beginners. We will also take a look at the Serial command available in the Arduino IDE. Module 2 covers reading and writing digital data. Module 3 progresses to reading and writing analog data along with working with interrupts. Module 4 will introduce servo-motor control using Pulse Width Modulation (PWM). Module 5 and 6 will allow members to put their newly acquired knowledge to test by building a mini-project that encompasses topics covered in previous modules.

The ultimate objective of this document and the Build-It-Yourself Shops is to get members equipped for better and bigger projects.

Contents

Module 4: Servomotors 1
Build! 2
Circuit 2
Code 3
Good to Know 5

Module 4: Servomotors

This module focuses on how to use a servomotor. A servomotor is a type of actuator that provides positional feedback. Essentially, a servomotor is a combination of a simple DC motor with a positional sensor and control circuitry. Most servomotors that we'll be working with use a potentiometer as the positional sensor. The potentiometer rotates along with the motor shaft and the potentiometer signal is read by the control circuit in the servomotor. The exercise at the end of this module will use a servomotor that intakes a value corresponding to the position of the servo, the control circuit moves the motor to the specified position based on feedback from the potentiometer.

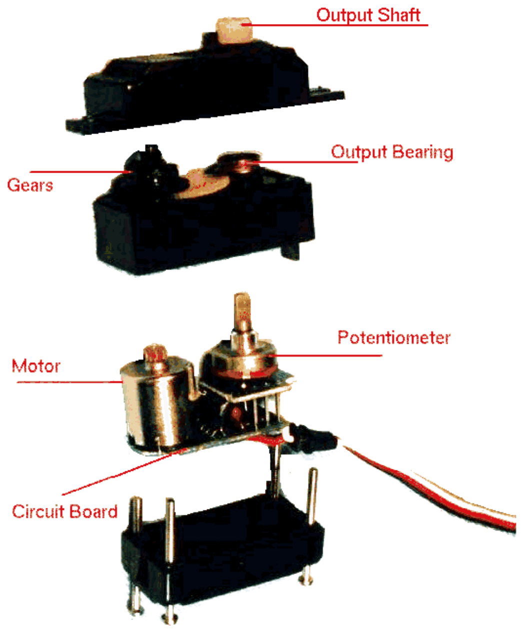


Figure 1: Components inside a typical hobby servomotor (Source: dm.risd.edu)

We're already familiar with how a potentiometer works. The voltage drop across the variables resistance helps the control circuit determine the direction of the DC motor shaft. Although we haven't worked with DC motors in any previous module, an analogy can be made between an LED and a simple DC motor. Just like an LED lights up when appropriate current flows through the terminals, a DC motor starts spinning when current flows through it. The DC motor stops spinning if there is no current, just like an LED would be off if there is no current. More voltage (more current) corresponds to increased motor speed, like higher brightness in the case of an LED. However, unlike an LED, a simple DC motor does not have any polarity. Hence, if voltage is applied in the opposite direction, then the motor will simply spin in the reverse direction.

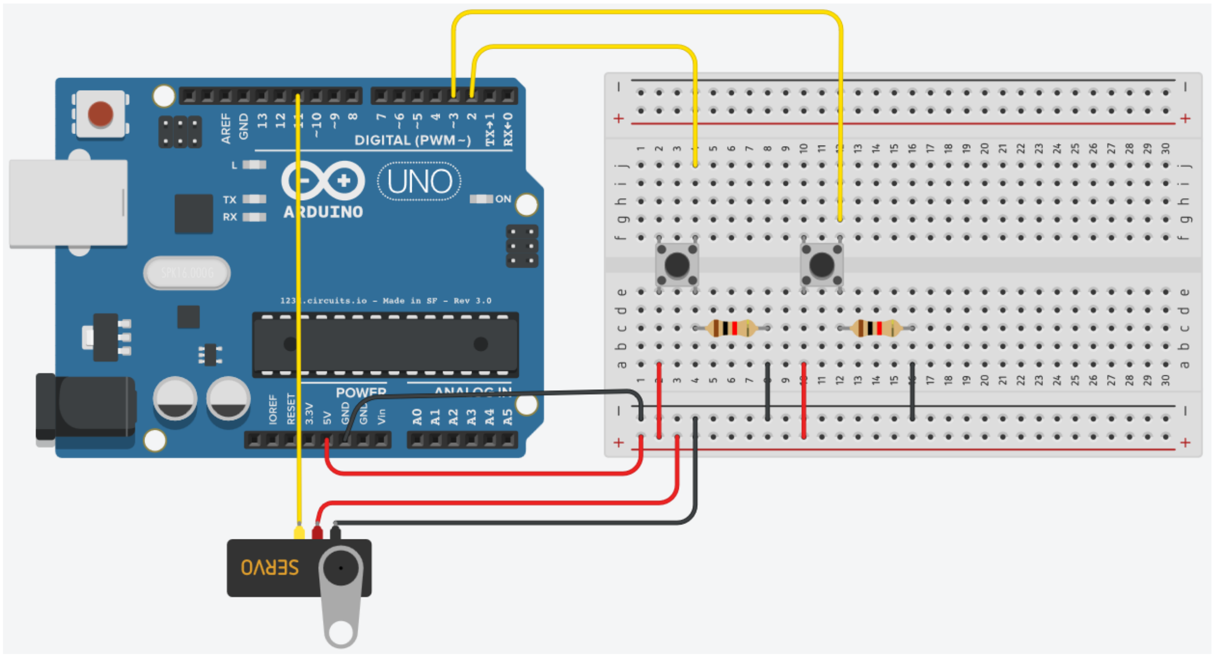
Build!

This building exercise involves changing the servomotor position based on input from pushbuttons.

Items required:

1. Microcontroller
2. 2 Pushbuttons
3. Resistors – 2 x 10k ohm
4. Servomotor

Circuit



Code

```
const int servoSignalPin = 11;

const int incButtonPin = 3;

const int decButtonPin = 2;

const int minPulseWidth = 1000;

const int maxPulseWidth = 2000;

const int medPulseWidth = 1500;

volatile int pulseWidth = medPulseWidth;


// the setup routine runs once when you press reset:

void setup() {

  // initialize the digital pin as an output.

  pinMode(servoSignalPin, OUTPUT);

  pinMode(incButtonPin, INPUT);

  pinMode(decButtonPin, INPUT);

  attachInterrupt(digitalPinToInterrupt(incButtonPin), incPulseWidth,
RISING);

  attachInterrupt(digitalPinToInterrupt(decButtonPin), decPulseWidth,
RISING);

  Serial.begin(9600);

}


// the loop routine runs over and over again forever:

void loop() {

  digitalWrite(servoSignalPin, HIGH);

  delayMicroseconds(pulseWidth);

  digitalWrite(servoSignalPin, LOW);
  delay(20);

  Serial.println(pulseWidth);

}


void incPulseWidth(){

  pulseWidth = pulseWidth + 100;

  if(pulseWidth >= maxPulseWidth){

    pulseWidth = maxPulseWidth;

  }

}


void decPulseWidth(){

  pulseWidth = pulseWidth - 100;

  if(pulseWidth <= minPulseWidth){

    pulseWidth = minPulseWidth;

  }

}
```

Good to Know

Communication Protocol

Here we can see the communication protocol used to set the angular position of the motor. For the servomotor shown in the figure (Parallax), the servomotor adjusts itself at 0 degrees for a pulse width of 1.5ms. For a pulse width of 2ms, the motor will adjust its angular position to 90 degrees. Similarly, for a pulse width of 1ms, the motor will move to an angular position of -90 degrees. For every pulse width between 1ms and 2ms, the control circuit will map the pulse width to angular position between -90 to 90 degrees. Notice that each HIGH pulse is followed by a LOW pulse of 20ms. This signal is received by the control circuit inside the servomotor, which spins the motor until the specified angular position is reached. Although the shown protocol here is from the Parallax datasheet, it holds true for almost any hobby servomotor.

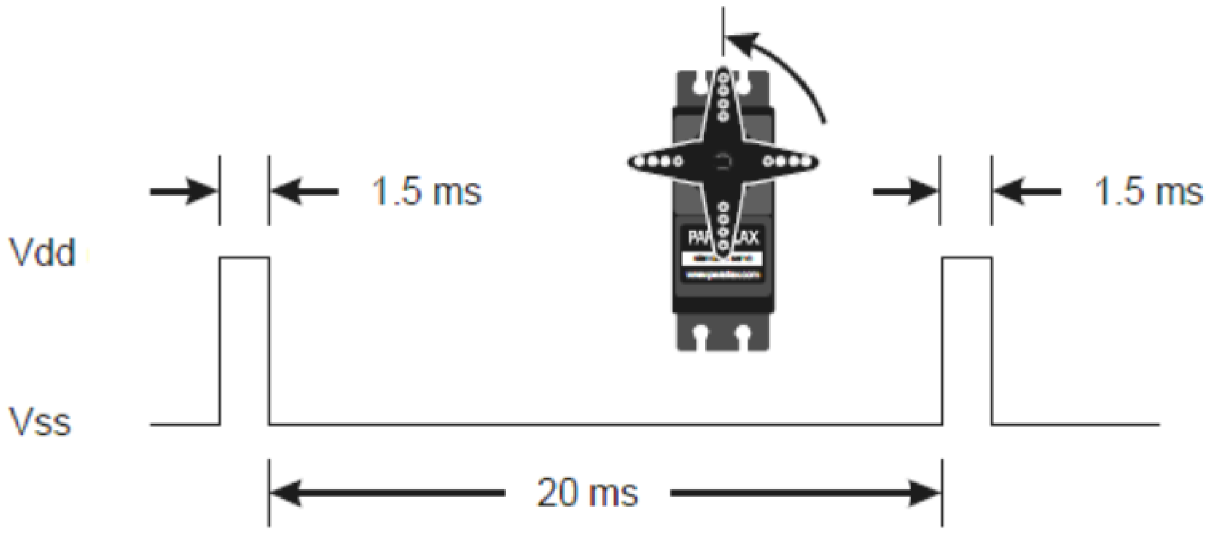


Figure 2: Communication protocol for a servomotor. (Parallax servomotor datasheet: goo.gl/BrVhl0)

Discontinuous Vs. Continuous Servomotor

This module introduces a servomotor which adjusts its angular position based on the pulse width of the signal. Sometimes these kinds of servomotors are referred to as discontinuous servomotors. These servomotors usually have a limited range of motion (~180^o), hence the name discontinuous. With a slight modification, we can convert these servomotors into ‘continuous’ servomotors, thereby allowing the motor shaft to spin continuously without any limitation. We know that the potentiometer acts as a positional sensor, therefore we can just simply cut the wire going from the potentiometer to the control circuit. This way the control circuit has no way to determine if the target angular position has been reached or not, which is why the control circuit will continuously instruct the motor to spin in the direction where the target position is. The further the target position, the faster the motor will spin. Therefore, by removing the positional sensing, the control circuit now become a speed controller instead of a position controller. Some servomotors may additionally require removal of a stop-pin on one of the gears inside the motor that physically limits the range of motion. Lots of examples for this type of conversion can be found through a simple web search.