ME 220 LAB: RC Servo Motor Control Using Arduino

Objective:
The objective of this lab is to get something moving by using the Arduino boards for the first time.

Pre-Lab:
Signals in the form of square waves are commonly used in mechatronic systems for several purposes. One application of such periodic signals is in motor actuation. In this lab we will focus on RC Servo motors (will be referred to as *RC Servo* in the remaining of this document). A RC Servo is not simply a DC motor. Indeed, it is a combination of a DC motor, a gear train, a position sensor (commonly a potentiometer is used) and control circuitry as illustrated in figure 1.

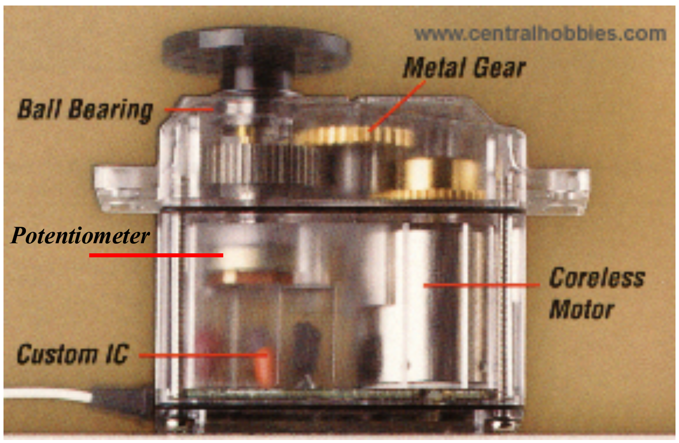


Figure 1. Components of a RC Servo [www.centralhobbies.com]

RC servos are small actuators widely used in remotely controlled (RC) toys (i.e. cars, planes, helicopters and boats). These are compact actuation mechanisms that provide an easy way of achieving position control in small scale applications where precision is not essential. Each RC servo is composed of a small DC motor attached to a gear train, a potentiometer and a simple controller. The gear reduction provides a reasonable amount of torque compared to their size. As the name implies, the RC servo has a sensor that tells the current position of the shaft to the controller that is built into this system, namely the potentiometer. The built-in controller compares the current position (measured from the pot) with the target (send from Arduino) and creates a control signal that will move the output shaft to a position that decreases this difference. By now you should have started wondering how we can set the reference position (in other words, how to tell the motor where to go). The reference position is provided by using a PWM (pulse width modulated) signal. All of the RC servos that you will find in our lab will have a range of 0-180^o. The built-in controller of RC servos control the rotor position within this range. Generally, one end of this range is referred to as –90^o and the other as +90^o (even though there is no harm if you prefer 0-180^o). Figure 2 illustrates the ideal working conditions of RC servos. As shown, RC servos can be rotated to specific positions by sending PWM signals, the ON (high) time of which determines the target position of the motor. In order to reach and maintain a position this PWM signal has to be **<u>continuously sent</u>** to the motor via its signal pin. Ideally high time of these signals will change between 1 to 2 ms. Low time of these signals can be anywhere between 10 to 25 ms. Use of 10 ms for low time is recommended. Even though ideal high times should be within [1-2]ms range, in practice this range will slightly change and in this lab you will determine these limits empirically.

Figure 2 illustrates the type of PWM that you have to generate in order to control the position of the RC Servo. Even though only three discrete positions are illustrated; the servos can be controlled continuously between –90 (P_{min}) and 90(P_{max}) degrees.

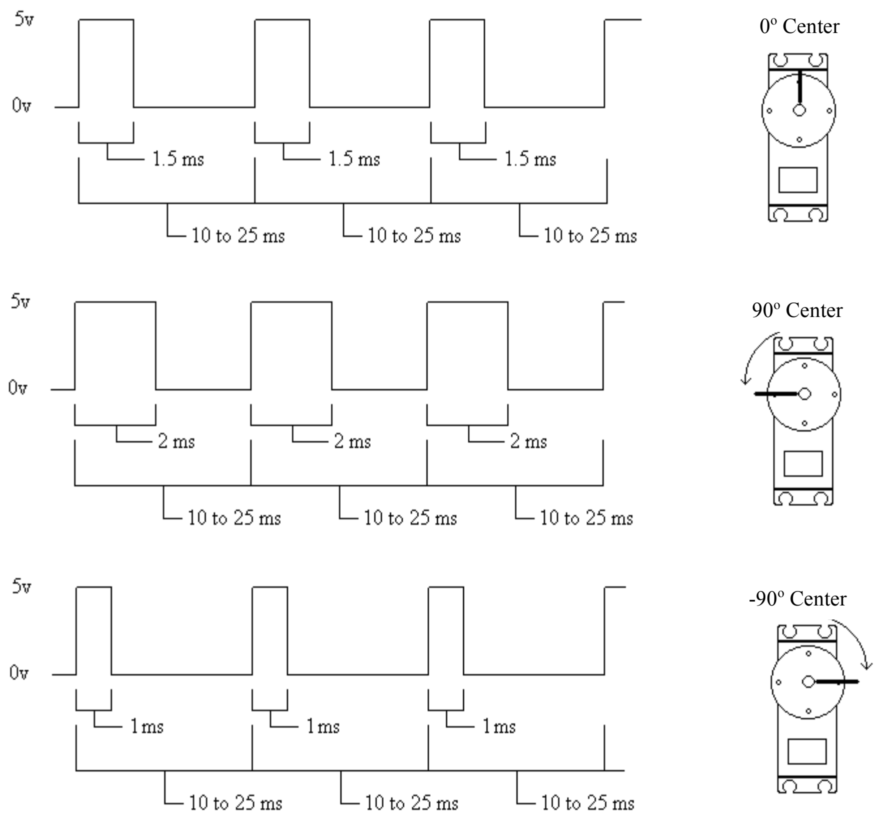


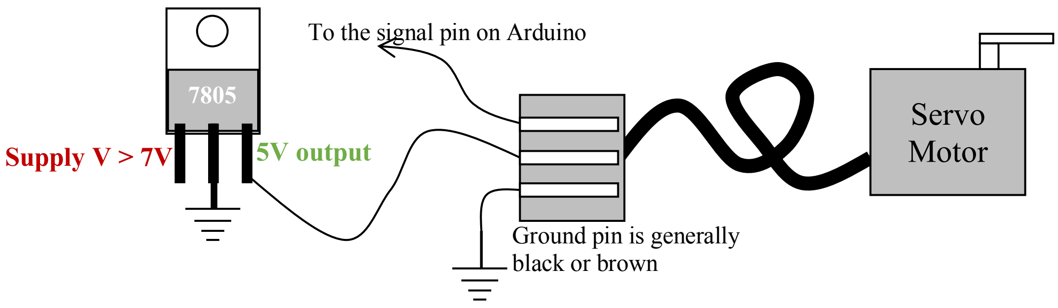
Figure 2. PWM Signal used in controlling a RC servo

Lab:

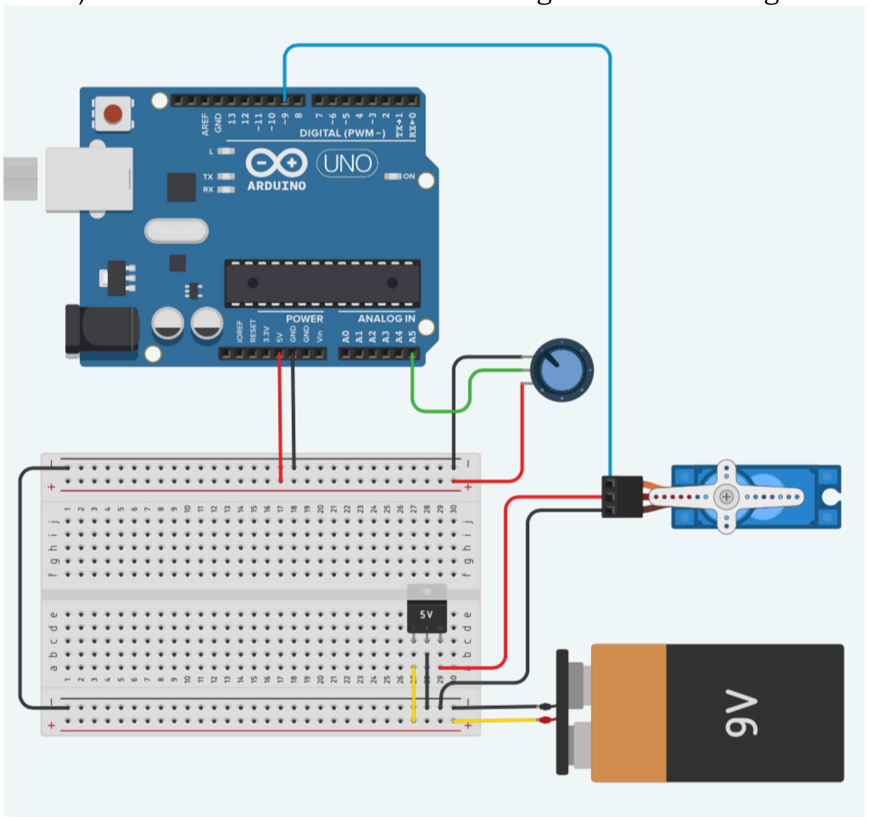
The lab will be completed in couple of steps.

1. Properly interface the RC Servo to the Arduino.
2. Write a RC servo control program on Arduino.
3. Interface a potentiometer (POT) to Arduino.
4. Implement a simple communication protocol with Arduino

1. Figure 2 illustrates the theoretical operation of RC servos. However, in practice, the actual values for RC servo limits might deviate from these theoretical ones. Connect the servo pins as follows: middle pin is 5V (supply wire is red in most motors), generally the darker colors such as black or brown are used to indicate ground, and the remaining pin is for signaling desired position.
Use of a separate regulated 5V source to feed the servo motor is absolutely recommended if a large size RC motor is used. In either case it is not a bad idea to prepare a regulated 5V supply using 7805. In that case make sure that *both sources have the same ground*. The proper wiring is shown in Figure 3. You can provide the signal from any digital output pin.


7805
To the signal pin on Arduino
Supply V > 7V
5V output
Servo Motor
Ground pin is generally black or brown

a) Schematic of RC servo interfacing to Arduino using 7805



b) Tinkercad example for RC servo interfacing to Arduino using 7805

Figure 3. How to interfacing the RC servo to Arduino using 7805 Voltage Regulator

2. Now, write a program in Arduino which receives a control position from the serial port, that is in the [-90 , 90] degrees range, and use this to command to motor to go to that position.

a) Converts user command, i.e. tha value in [-90 , 90] range to a signal as explained in Figure 2 and **continuously** applies this signal to the RC Servo pin. Note that as soon as you stop sending this signal, the motor is *released.* The details of this messaging protocol are up to you, but if you send anything outside the range [-90 , 90], Arduino should stop sending PWM signals to the motor and hence motor should be *released*.
What to use:`digitalWrite(), delayMicroseconds()`
b) Do the same using the Arduino library for servos. Check out the servo library from Arduino webpage for more details.
What to use: `#include <Servo.h>, attach(), write(), detach()` functions from Servo library.

3. Next step is to control the RC servo using your POT. Your program should map POT output to the RC servo position, and as you turn the POT from one end to the other, RC servo should turn in sync with the POT.

4. Final step is to control the RC servo through a *communication protocol.* Through this protocol you will be able to send Arduino **complex motion patterns**. A message in this protocol starts with letter 's' and ends with letter 'e'. Between 's' and 'e' there will be pairs of position and duration (`pos:wait`). Each (`pos:wait`) pair will be separated with commas. 'pos' refers to position in degrees in [-90,90] range and '`wait`' is amount of delay in miliseconds. There may be *any number* of (`pos:wait`) pairs in a message, yet anything more than 10 is not very practical.
A sample message is as follows:

s45:1000,-45:1000,0:2000e

Your code can start running as soon as it receives the first valid pair, or when the message is fully complete. It is up to you. After the last command (i.e. the one right before 'e') is executed, the motor should be released.