Lab 5: Servo Control with PWM

Name: Shahmir Mustafa Chaudhry ID: 08400 Section: T1

Objective

To control SG90 Servo with TivaC Launchpad using PWM Generation and interface with Ultrasonic sensor.

In-Lab Task:

Task 1: Generate 50Hz PWM signal to control position of servo

Task 2: Using integrated Servo library and control with push button

Task 3: Incorporate Servo to Critical Distance Measurement System

1 Introduction to SG90 Servo

SG90 is a low cost and high output power servo motor. It can not move continuously but rotates up to 180 degrees and each step can be of maximum 90 degrees. Moreover, it is small enough that it can easily fit into your robotics ARM or obstacle avoidance robotics projects. On top of that, it requires only one output pulse signal to control its movement.

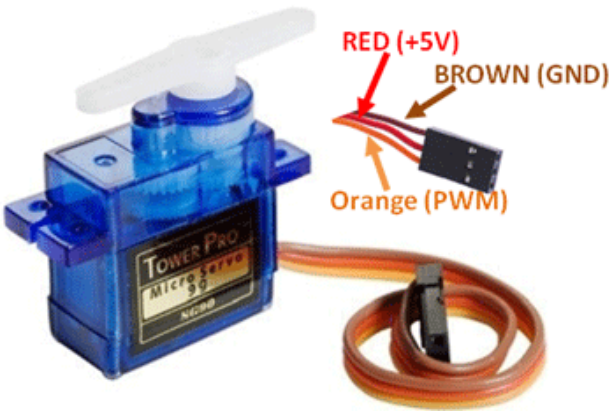


Figure 1: SG90 Servo Pins

Fig. 1 shows the pinout diagram of SG90 servo motor. It consists of three pins only such as PWM, ground and Vcc pin. Brown, orange and red wires are GND, Vcc and PWM pins respectively.

Vcc and ground, as their name suggests, are the power supply pins which are used to power servo motor. Only 5 volts of power signal is required to power this motor. Mostly microcontrollers or development boards have onboard 5 volts supply which we can use to power SG90 servos. PWM pin is used to provide specific duty cycle signal to control the position of servo.

1.1 Angular Position Control for Servo SG90

A servo motor responds to changes in duration of pulses. To control rotation, we apply pulses to the PWM pin of the servo motor. The frequency of the PWM signal should be 50Hz for SG90 Servo (mentioned in datasheet of SG90). That means the control signal pulse should be applied to the PWM pin every 1/50Hz = 20ms duration. The width of the pulse has a direct relation with the angular position of the motor. The pulse width may vary depending for different servo model. Unlike a typical square waveform where in the signal is high for half of the period (and low for the other half), we seek to control the "duty Cycle" (modulate the width of the pulse) of the waveform by varying the interval over which the signal is high (T_{ON}). Note that $T = T_{ON} + T_{OFF}$.

For SG90, following pulse width helps us achieve the desired angular position and as shown in Fig. 2:

- Pulse of 1 ms makes it rotate all the way to the left (-90 degrees)
- Pulse of 1.5 ms makes the motor return to the middle (0 degrees)
- Pulse of 2 ms makes it rotate all the way to the right (90 degrees)

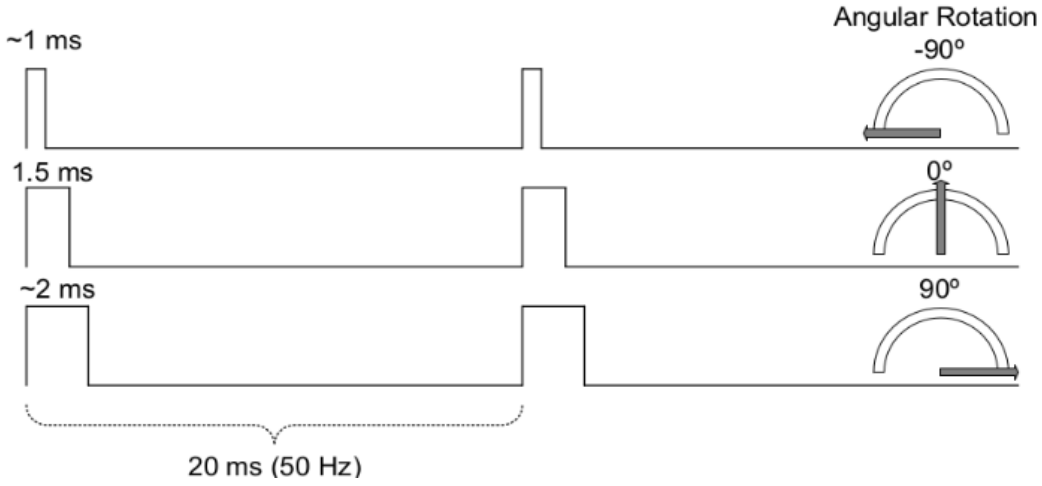


Figure 2: Position Control Duty Cycle for SG90

2 TivaC LaunchPad with Energia

In Energia IDE, unlike Kiel uVision, we can use TivaC LaunchPad pins for various peripherals without the need to activate Ports using registers or specifying function of the pin. But it also comes with limitations of usability and programmable scope of the board. In Energia, we can refer to Pins of TivaC directly using numeric digit like 1,2,3.. and so on. Pin map for the EK-TM4C123GXL LaunchPad is given in Fig. 3 with Black Columns under J1, J2, J3 and J4. In this lab, we will see the limitation of using Energia for programming our device primarily by accessibility to hardware peripherals.

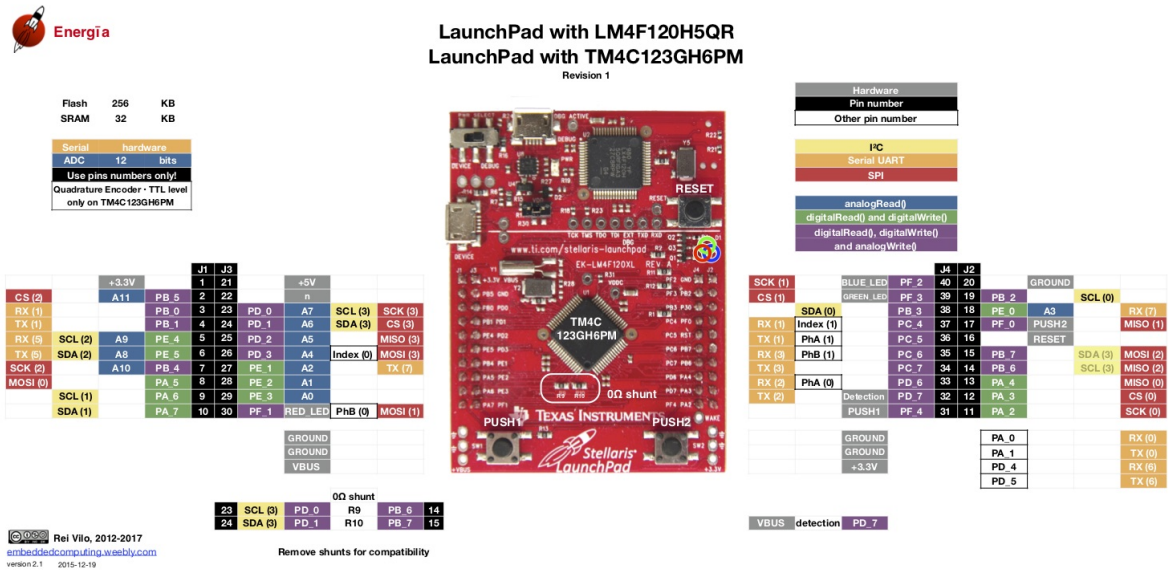


Figure 3: Pin Configuration of TivaC for Energia (Front)

3 In-Lab Tasks

Task 1: Generate 50Hz PWM signal to control position of servo

In order to use the SG90 with TivaC, follow the steps given below:

1 Make the pin connections between TivaC, SG90 and Power Supply using Table. 1 by using Power Supply 5V fixed terminal to provide 3.3V to servo. You can also use Fig. 4 to see how we provide power to our servo from PSU. **DO NOT USE 3.3V OF TIVA-C TO POWER SERVO AS IT MAY DAMAGE IT**.

2 Download the file "servo_ sonic.ino" from LMS and load the code in Energia IDE, go through each line carefully to understand each operation as you will be writing your own code for later tasks and later labs.

3 Write your logic in code where you are asked to i.e. "servo_ p90_ Degree()" and "servo_ m90_ Degree()" function as per explanation given.

4 Connect tivaC, compile/build and download code on tivaC.

Table 1: Task01 Connections

SG90	**TivaC**	**Energia**	PSU
Vcc (Red)	-	-	3.3V
PWM (Orange)	PF1	30	-
GND (Brown)	GND	-	GND

You should be able to notice huge error margin in the angular displacement achieved via PWM generated using delayMicroseconds(), this primarily happens due to mentioned reasons below:

- We are not providing exact 50Hz signal as that would require additional tuning of code as per the clock frequency of TivaC
- SG90 Servo encoder responsible for measuring duty cycle isn't efficient as it is a basic servo
- The duty cycle estimated duration i.e. 1ms, 1.5ms and 2ms are approximations
- delayMicroseconds() have certain margin for error as it halt the code execution altogether

In order to cater the above mentioned constrains, Eneregia provides libraries such as "Servo.h" that uses hardware timers to generate PWM which are far more accurate and precise than the PWM generated by delayMicroseconds(). We will use this library to efficiently control servo position in next task by achieving any angular position between range of 0 to 180 degrees instead of only 3 positions. You will also learn the implementation of hardware timers in later lab.

*Provide **screenshot** of code where you added your logic, add in as many boxes as necessary below. Get the circuit demonstration checked with RA within Lab. Make sure the code is readable.*

```
//////////////////////////////////////////////////
void servo_0_Degree(){
    //We use delayMicroseconds() instead of delay(ms)...
    //...since delayMicroseconds() are more precise and give accurate PWM

    // Rotates servo to 0 degree, total pulse duration is 20ms
    digitalWrite(Servo_control_pin, HIGH);
    delayMicroseconds(1500); // High for 1.5ms, Duty Cycle of 7.5%
    digitalWrite(Servo_control_pin, LOW);
    delayMicroseconds(18500); // Off for remaining  18.5ms
}

void servo_p90_Degree(){
    // Rotates servo to +90 degree, total pulse duration is 20ms
    // Write your logic here for getting +90 degrees

    // Rotates servo to 90 degree, total pulse duration is 20ms
    digitalWrite(Servo_control_pin, HIGH);
    delayMicroseconds(1000); // High for 1ms, Duty Cycle of 5%
    digitalWrite(Servo_control_pin, LOW);
    delayMicroseconds(19000); // Off for remaining  19ms
```

```
void servo_m90_Degree(){
    // Rotates servo to -90 degree, total pulse duration is 20ms
    // Write your logic here for getting -90 degrees

    // Rotates servo to -90 degree, total pulse duration is 20ms
    digitalWrite(Servo_control_pin, HIGH);
    delayMicroseconds(2000); // High for 2ms, Duty Cycle of 10%
    digitalWrite(Servo_control_pin, LOW);
    delayMicroseconds(18000); // Off for remaining  18ms
}
```

Task 2: Using integrated Servo library and control with push button

For this task, we will use the built-in "servo.h" library to control the Servo Position with far greater accuracy as the library use the hardware timer and allows us to set position for any any number between range of 0 to 180 degrees. You are required to rotate servo to and fro from 0 to 180 degrees whenever switch SW2 onboard is pressed. For this purpose, perform the following steps:

1 Uncomment the commands in "servo_ sonic.ino" under Task2 headings wherever present.

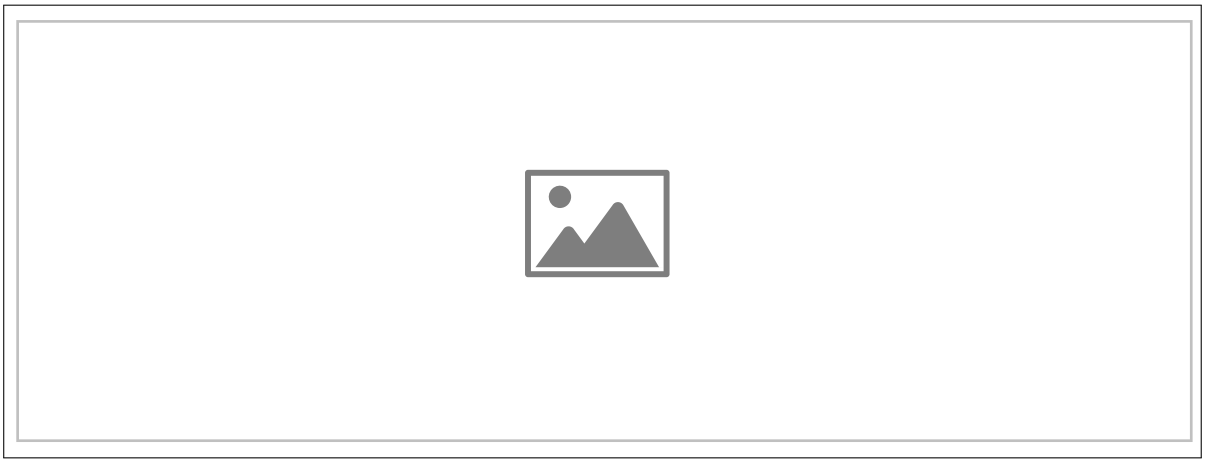
2 Write your logic in code where you are asked to i.e. "Loop to make servo go from 180 degrees to 0 degrees".

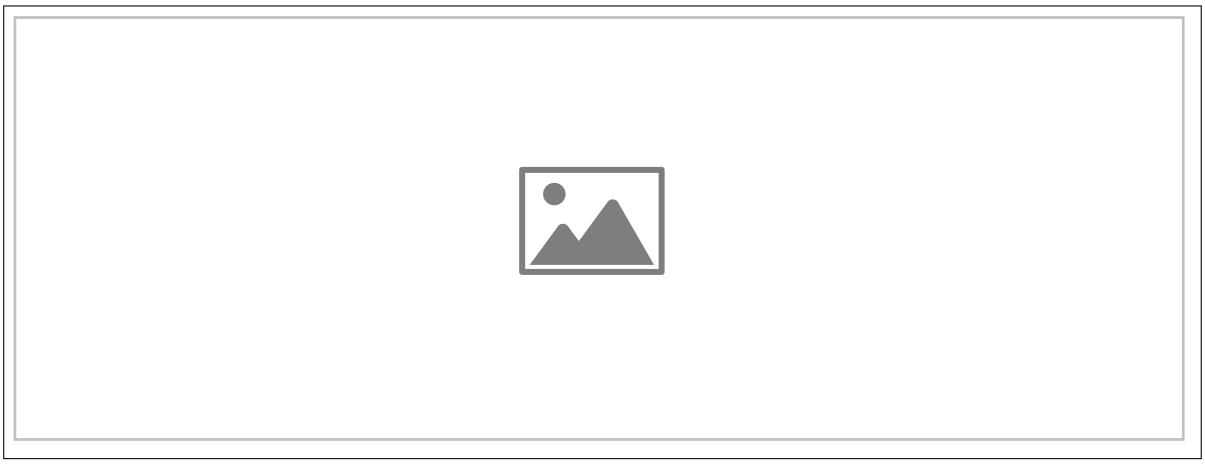
3 Test your code first to ensure it works without push button SW2 i.e. continuous rotation from 0 to 180 degrees and from 180 degrees back to 0 degrees.

4 Once done, utilize SW2 state, such that whenever the SW2 is pressed, the servo rotates to and fro, otherwise it should be static and still. Whenever SW2 is pressed, (digitalRead(SW2) == 0) is true.

*Provide **screenshot** of code where you added your logic, add in as many boxes as necessary below. Get the circuit demonstration checked with RA within Lab. Make sure the code is readable.*

```
// */

//Task 02: Using <servo.h> library to control angular position and add SW2
///*
  //Loop here that make servo goes from 0 degrees to 180 degrees
  if (digitalRead(SW2) == LOW){
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(5);                        // waits 5ms for the servo to reach the position
  }


  //Write a Loop here that make servo goes from 180 degrees to 0 degrees
  for(pos = 180; pos > 0; pos -= 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(5);                        // waits 5ms for the servo to reach the position
  }
  }
//*/
```





Task 3: Incorporate Servo to Critical Distance Measurement System

Program you TivaC board to design a "Critical Distance Measurement System with Servo Control" by using the Table in Table. 2 to decide circuit connections and using Servo.h library functions. The system should contain Ultrasonic Sensor interfaced with Servo using TivaC. The system should fulfill the following requirement:

1. Servo at 45 degrees for range of 0-15cm
2. Servo at 90 degrees for range of 15-30cm
3. Servo at 180 degrees for range of 30-45cm
4. For any other case, Servo at 0 degree

Table 2: Task02 Connections

HC-SR04	**TivaC**	**Energia**	**Power Supply**
Vcc	-	-	3.3V
Trig	PA7	10	
Echo	PA6	9	
GND	GND	-	GND
SG90	-	-	
Vcc (Red)	-	-	3.3V
PWM (Orange)	PF1	30	
GND (Brown)	GND	-	GND
-	SW2	17	

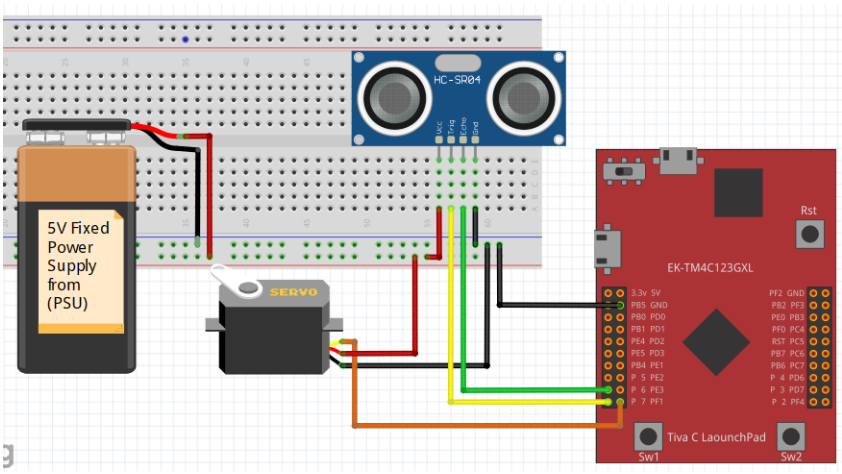
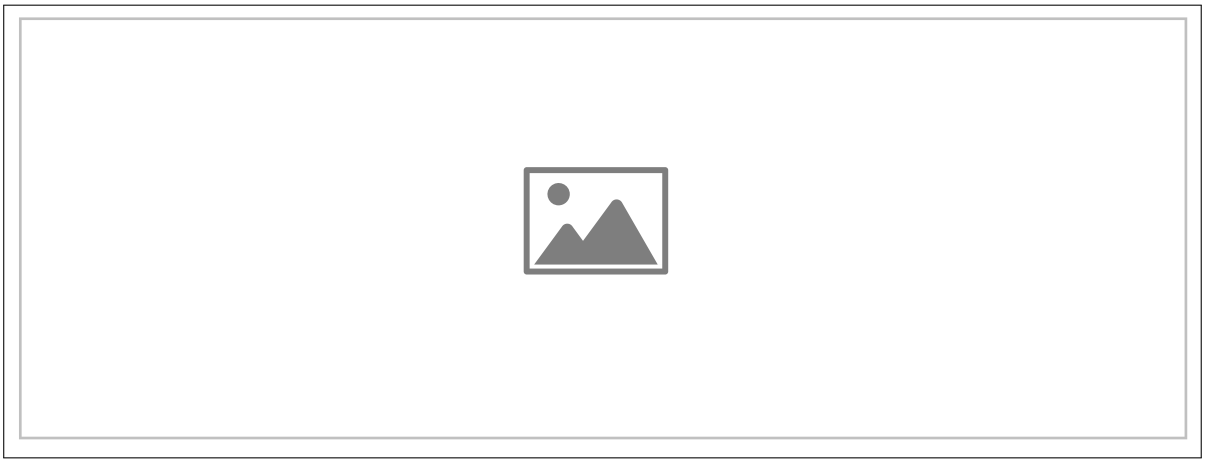
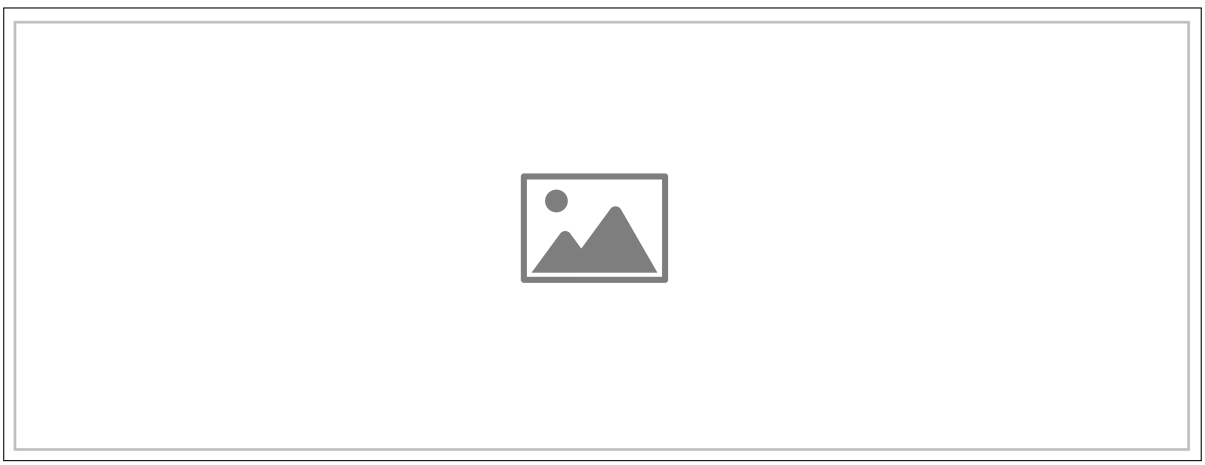


Figure 4: Pin Connection of TivaC with Ultrasonic Sensor and Servo

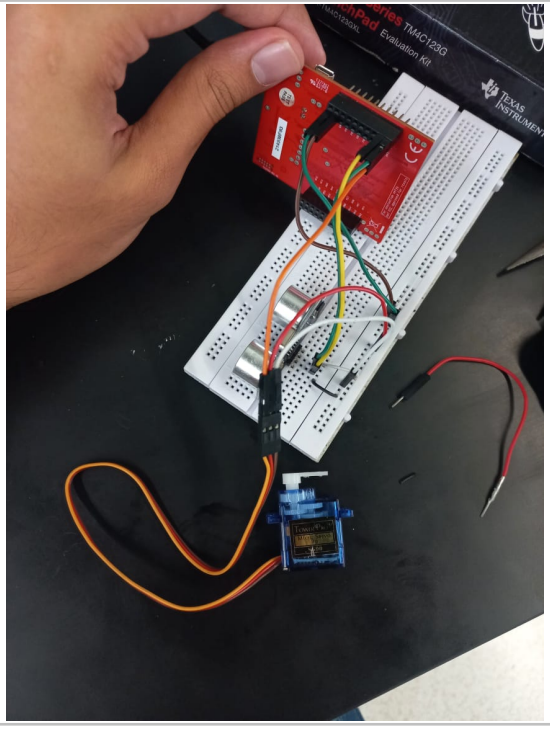
*Provide **screenshot** of code where you added your logic, add in as many boxes as necessary below. Get the circuit demonstration checked with RA within Lab. Make sure the code is readable.*

```
//Task 03: Interfacing servo with ultrasonic sensor

 Update_Ultra_Sonic(); //Update distance from ultrasonic sensor
 delay(500);

//Write your logic for task03 here, use "cm" to read distance
if (cm >0 && cm<15){
  for(pos = 0; pos < 45; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                 // in steps of 1 degree
    myservo.write(pos);             // tell servo to go to position in variable 'pos'
    delay(5);                       // waits 5ms for the servo to reach the position
  }
  }

else if (cm >15 && cm<30){
  for(pos = 0; pos < 90; pos += 1)  // goes from 0 degrees to 90 degrees
  {                                 // in steps of 1 degree
    myservo.write(pos);             // tell servo to go to position in variable 'pos'
    delay(5);                       // waits 5ms for the servo to reach the position
  }
  }
else if (cm >30 && cm<45){
  for(pos = 0; pos < 180; pos += 1)  // goes from 0 degrees to 180 degrees
  {                                  // in steps of 1 degree
    myservo.write(pos);              // tell servo to go to position in variable 'pos'
    delay(5);                        // waits 5ms for the servo to reach the position
  }
  }
 else{
    myservo.write(0);              // tell servo to go to position in variable 'pos'
    delay(5);
 }
```





Provide your clear circuit image below



4 Assessment Rubrics

Marks distribution

		LR2	LR4	LR5	LR9
In-lab	Task 1	5 points	-	5 points	10 points
	Task 2	10 points	10 points	10 points	
	Task 3	10 points	20 points	20 points	
Total Marks 100					

Marks obtained

		LR2	LR4	LR5	LR9
In-lab	Task 1		-		
	Task 2				
	Task 3				
Obt. Marks out of 100					