Practical 6: Programs using Servo Motors

Aim:

To control the motion of a Servo motor using Arduino

Simulation Environment: TinkerCAD (Free online simulator)

Components: Arduino UNO, Servo motor

Theory:

A micro servo motor is a small-sized servo motor designed for applications where space is limited. Servo motors, in general, are devices that incorporate a feedback mechanism to control the speed and position of the motor accurately. They are commonly used in robotics, remote- controlled vehicles, and various other projects where precise control of movement is required.

A micro servo motor functions as follows:

Motor: The motor inside the servo is responsible for producing the mechanical motion. It typically consists of a DC motor.

Gear Train: Servos have a gear train that converts the high-speed, low-torque output of the motor into low-speed, high-torque motion.

Control Circuitry: The control circuitry is responsible for interpreting the signals received from an external source (like an Arduino) and translating them into precise movements.

Potentiometer (Feedback Device): Most servo motors have a potentiometer (a variable resistor) connected to the output shaft. This potentiometer provides feedback to the control circuitry about the current position of the motor.

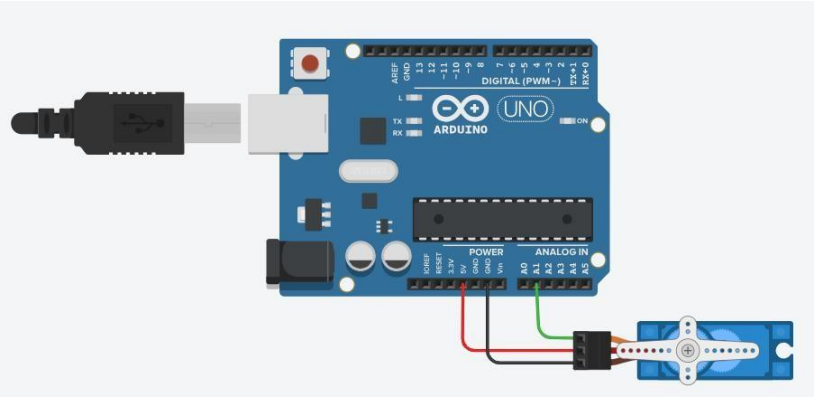
When we connect a micro servo motor to an Arduino, we typically use a library (such as the Servo library in Arduino) to control its movements.

The working of Servo motor interfaced with Arduino can be understood as follows

The movement of a servo motor attached to an Arduino is controlled by sending a series of pulses to the servo motor. These pulses are typically generated using a technique called Pulse Width Modulation (PWM).

a. Pulse Width Modulation (PWM): Arduino boards have digital pins that can output PWM signals. PWM is a technique where the duration of a pulse is varied while the frequency remains constant. In the case of servo motors, the pulse width is crucial because it determines the position to which the servo motor should move.
b. Servo Library: Arduino provides a Servo library that simplifies the task of controlling servo motors. This library abstracts the details of generating PWM signals, making it easier to control the servo.
c. Attach Function: In the Arduino code, you first use the `attach` function to associate a servo object with a specific pin on the Arduino to which the signal wire of the servo is connected.
d. Write Function: To move the servo to a specific position, you use the `write` function. The argument passed to this function is the desired angle. The angle corresponds to the position to which the servo should move. For example, `myservo.write(90);` would move the servo to the 90-degree position.
e. Pulse Generation: Internally, the Servo library translates the angle specified in the `write` function into an appropriate pulse width. The library generates the necessary PWM signal, and the Arduino outputs this signal through the specified digital pin.
f. Control Loop: The servo motor's control circuitry interprets the PWM signal and adjusts the position of the motor accordingly. The feedback mechanism (potentiometer) inside the servo constantly provides information about the motor's current position to ensure that it reaches and maintains the desired position.
g. Looping or Sequential Control: In a loop or sequence of commands, you can vary the angles sent to the servo to make it move continuously or in a specific pattern.

Circuit Diagram:



Pin Connections:

Arduino	Servo Motor
5V	Power
GND	Ground
Pin A_1	Signal

Code:

```
#include <Servo.h>
Servo servoBase; // Create a Servo object and assign it a specific name
void setup() {
servoBase.attach(A1); // Specify the pin to use for the servo
servoBase.write(0); // Set the servo motor to the 0-degree position
}
void loop() {
 // Sweep the servo from 0 to 180 degrees in steps of 10 degrees
for (int i = 0; i <= 180; i += 10) {
servoBase.write(i); // Set the servo to the current angle
delay(2000); // Pause for 2000 milliseconds (2 seconds)
 }
}
```

Practical 7: Programs using digital infrared motion sensors

Aim:

To detect motion of any object using Infrared sensors

Simulation Environment: TinkerCAD (Free online simulator)

Components: Arduino UNO, Passive Infrared (PIR) Sensor, LED and resistor (1 KΩ)

Theory:

A Passive Infrared (PIR) sensor is a type of electronic sensor that detects infrared (IR) radiation emitted by objects in its field of view. PIR sensors are often used to detect motion and are commonly found in security systems, lighting control, and other applications where the presence of people or animals needs to be detected.

Here's how a basic PIR sensor works:

1. Detection of Infrared Radiation: PIR sensors are equipped with a special material that is sensitive to infrared radiation. When an object with a temperature above absolute zero (- 273.15°C or -459.67°F) moves in the sensor's field of view, it emits infrared radiation.

2. Pyroelectric Material: The sensor contains a pyroelectric material, typically a crystal that generates a voltage when exposed to changes in temperature. The pyroelectric material is divided into segments, and each segment is connected to a pair of electrodes.

3. Detection of Changes in Infrared Radiation: As an object moves within the sensor's detection range, the amount of infrared radiation reaching different segments of the pyroelectric material changes. This results in variations in the voltage generated by the material.

4. Signal Processing: The sensor's electronics process these voltage changes and convert them into a signal that indicates motion or the presence of a heat source.

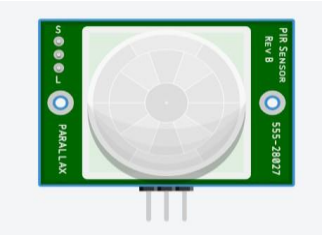
5. Output Signal: The sensor typically provides a digital output signal that can be used to trigger an alarm, turn on lights, or perform other actions based on the detected motion.

One of the key advantages of PIR sensors is their low cost, simplicity, and efficiency in motion detection applications. However, it's essential to note that PIR sensors can be sensitive to changes in temperature and may produce false alarms in certain situations, such as when there are sudden temperature changes in the environment. Advanced PIR sensor designs and signal processing techniques are employed to minimize false positives and enhance overall performance.

All objects (having temperature higher than absolute zero) emit radiations from the generated heat. These radiations cannot be detected by a human eye. Hence, electronic devices such as motion sensors, etc. are used for the detection of these radiations.
The advantages of using a PIR sensor are listed below:

a) Inexpensive
b) Adjustable module
c) Efficient
d) Small in size
e) Less power consumption
f) It can detect motion in the dark as well as light.

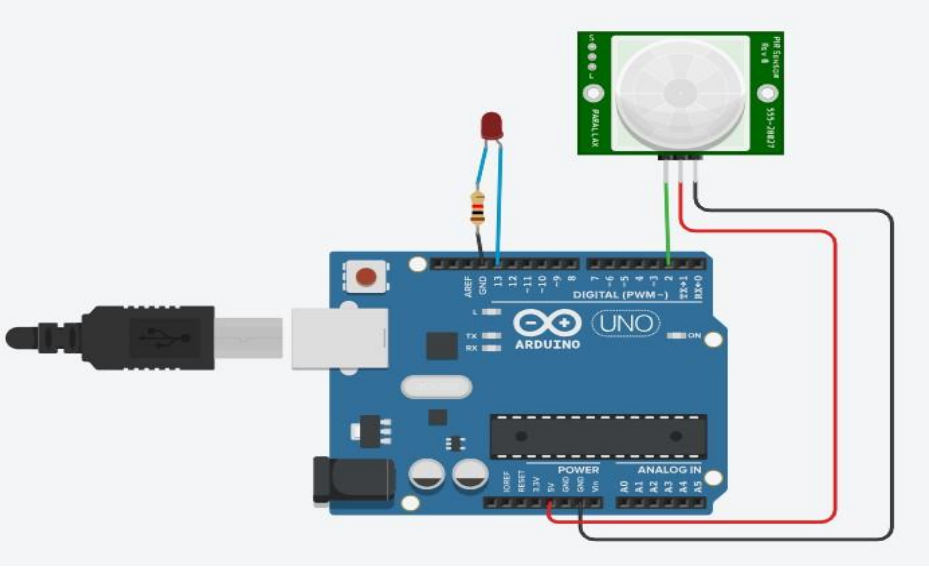
The PIR sensor has three terminals, which are listed below:



a) VCC
b) Digital Output
c) GND (Ground)

We will connect the Vcc terminal of the sensor to the 5V on the Arduino board. The PIR's sensor output can be connected to any of the digital pins on the Arduino board.The detection range of PIR sensors is from 5m to 12m.

Circuit Diagram:



Pin Connections:

Arduino	PIR Sensor	LED
5V	Power	
GND	Ground	
Pin 2	Signal	
GND (Digital)		Cathode through the resistor
Pin 13		Anode

Code:

```
// C++ code //
int sensorState = 0; void setup()
{
pinMode(2, INPUT); pinMode(LED_BUILTIN, OUTPUT);
}
void loop()
{
// read the state of the sensor/digital input
  sensorState = digitalRead(2);
// check if sensor pin is HIGH. if it is, set the LED on.
  if (sensorState == HIGH) { digitalWrite(LED_BUILTIN, HIGH);
}
else
{
digitalWrite(LED_BUILTIN, LOW);
}
delay(10); // Delay a little bit to improve simulation performance
}
```