SYIT

Embedded Systems Practical file

Ismail H. Popatia

Assistant Prof.

Computer Science Dept.

Maharashtra College

INDEX

Practical 1: Introduction to Arduino

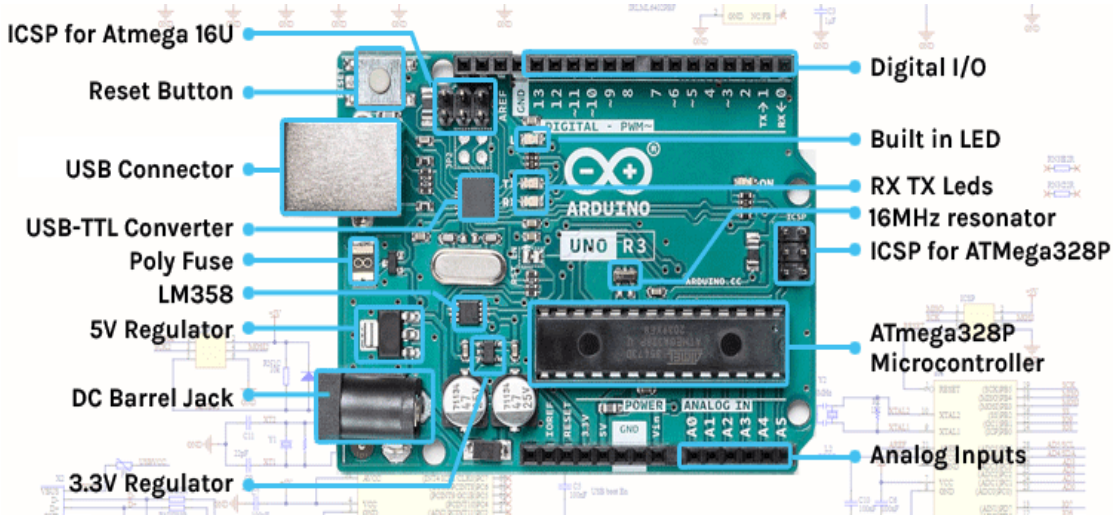
Aim:

1. To study the basics of Arduino circuits and bread-boarding
2. Blinking of LEDs

Simulation Environment: TinkerCAD (Free online simulator)

Part A: Basics of Arduino Circuits

Theory:



Arduino is an open-source electronics platform that has gained immense popularity for its ease of use and versatility. It was created in 2005 by a group of Italian engineers and is now maintained and developed by the Arduino community.

The heart of the Arduino platform is a microcontroller, which is a small, programmable computer on a single integrated circuit (IC) chip.

Arduino boards, which house these microcontrollers, provide a user-friendly environment for creating interactive electronic projects, prototypes, and various applications.

Key Components of Arduino:

1. Microcontroller: The core of an Arduino board is the microcontroller. The most commonly used microcontroller in Arduino is the ATmega series from Atmel (now a part of Microchip Technology). These microcontrollers come in different variations and are the brains behind your Arduino projects.

2. Input/output Pins: Arduino boards have a set of digital and analog pins that can be used to read data (inputs) or send data (outputs). Digital pins work with binary signals (0 or 1), while analog pins can read a range of values. The number and types of pins vary among different Arduino board models.

3. Power Supply: Arduino boards can be powered via USB, an external power supply, or a battery. Some boards have built-in voltage regulators, which make them compatible with a range of power sources.

4. USB Port: Arduino boards often feature a USB port for programming and power supply. This allows you to connect the board to your computer and upload code.

5. Reset Button: A reset button is provided to restart the Arduino, allowing you to upload new code or reset the program.

6. LED Indicator: Many Arduino boards include a built-in LED (Light Emitting Diode) on pin 13, which can be used for testing and basic visual feedback.

Arduino Software:

The Arduino platform comes with its integrated development environment (IDE). The Arduino IDE is a software tool that allows you to write, compile, and upload code to the Arduino board. Key features of the IDE include:

- Programming Language: Arduino uses a simplified version of the C/C++ programming language. It provides a set of libraries and functions tailored for easy interaction with the hardware.

- Code Library: Arduino has a vast library of pre-written code and functions that simplify common tasks, making it accessible to beginners.

- Serial Monitor: The IDE includes a serial monitor that allows you to communicate with the Arduino board and view debugging information.

- Community Support: The Arduino community is large and active, offering forums, tutorials, and extensive documentation to help users troubleshoot issues and learn.

Part B: Blinking of LEDs

Components Used:

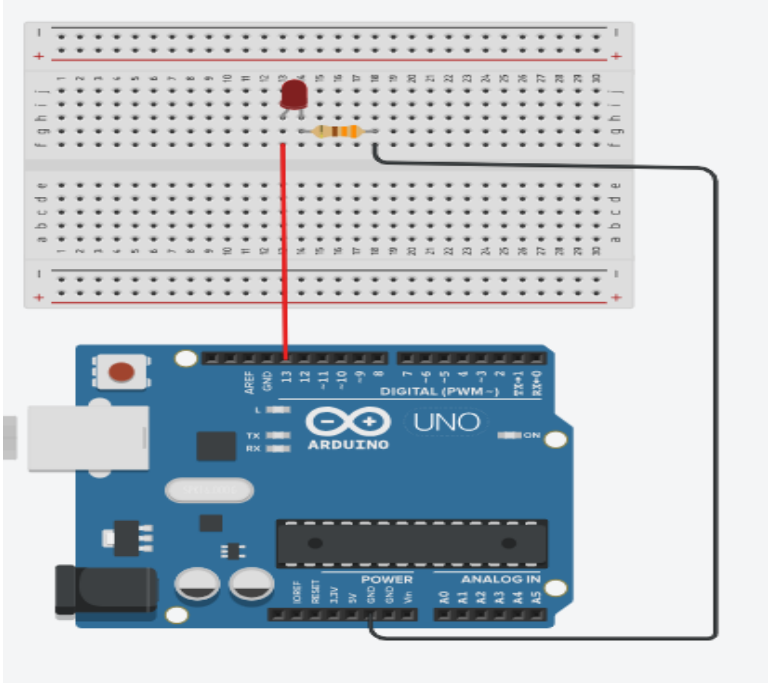
1. Arduino UNO
2. Breadboard
3. LED
4. Resistor (330 Ω)

The Following is the Circuit diagram we need to implement using the TinkerCAD simulation environment,

Arduino UNO is used to blink the LED continuously, we connect the pin 13 to the anode of the LED and cathode of the LED is connected to a resistor (330 Ω) ro limit the current passing through the LED. If large current flows through the LED then it may damage the LED (in real world environment).

The other end of the LED is terminated to the ground connection of the Arduino to complete the circuit.

Circuit Diagram:



Code: The following C++ code is used in the given case

```
void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  digitalWrite(LED_BUILTIN, HIGH);
  delay(1000); // Wait for 1000 millisecond(s)
  digitalWrite(LED_BUILTIN, LOW);
  delay(1000); // Wait for 1000 millisecond(s)
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/cCFZGNqm9EY

Practical 2: Program using Light Sensitive Sensors

Aim:

To study the working of Light sensor using Arduino

Simulation Environment:

TinkerCAD (Free online simulator)

Components:

Arduino UNO, LED, Photodiode and Resistors

Theory:

The goal of this practical is to create a system that can automatically control the brightness of an LED based on the light detected by a photodiode. This project leverages the principles of light sensing and feedback control.

Components:

a) Photodiode: A photodiode is a light-sensitive semiconductor device that generates a current or voltage proportional to the incident light's intensity. It acts as the input sensor in this system.
b) LED: An LED (Light Emitting Diode) is used as the output device. It emits light and can be controlled to vary its brightness.
c) Arduino: The Arduino microcontroller is the brain of the project. It reads data from the photodiode, processes it, and controls the LED's brightness accordingly.

Working:

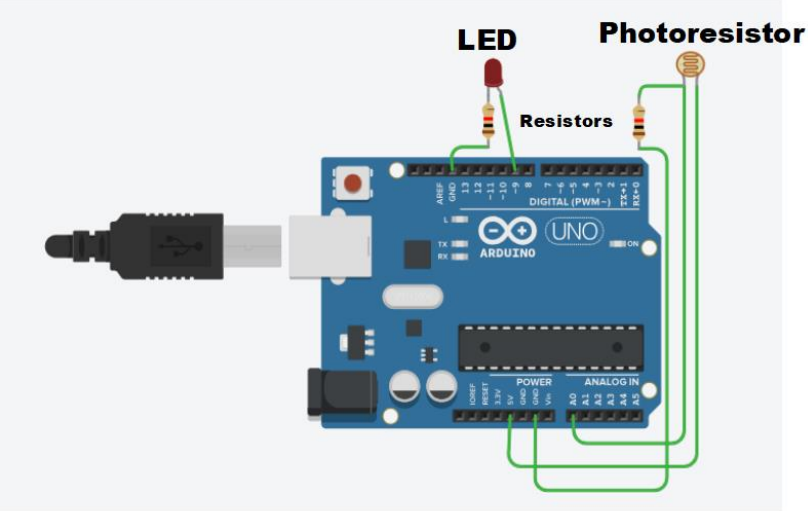
a) Photodiode Operation:
The photodiode is connected to one of the Arduino's analog input pins.
When exposed to light, the photodiode generates a current or voltage that is directly proportional to the light intensity.
Arduino reads the analog voltage from the photodiode using one of its analog pins.

b) Control Algorithm:
The Arduino is programmed with an algorithm that translates the analog reading from the photodiode into a control signal for the LED.
The algorithm typically involves mapping the photodiode's output to the LED's brightness. For example, when the photodiode detects more light, the LED becomes brighter, and vice versa.

c) Feedback Loop:
The system operates in a feedback loop. As light conditions change, the photodiode detects the variations and sends this information to the Arduino.
The Arduino processes the data and adjusts the LED's brightness in real-time based on the input from the photodiode.

This closed-loop system ensures that the LED's brightness is always synchronized with the surrounding light levels.

Circuit Diagram:



Pin Connections:

Arduino	Photoresistor	LED
5V	Right pin	
GND (Power)	Left pin through a Series Resistor	
A_0	Left pin	
Pin 9		Anode
GND (Digital)		Cathode through a Series Resistor

Code: The following C++ code is used in the given case

```
intlightSensorValue = 0; // Variable to store the light sensor reading
void setup() {
pinMode(A0, INPUT);   // Set A0 pin as an input for the light sensor
pinMode(9, OUTPUT);   // Set pin 9 as an output to control the LED
Serial.begin(9600);  // Initialize serial communication at 9600 baud
}
void loop() {
```

```
lightSensorValue = analogRead(A0);      // Read the light sensor value
Serial.println(lightSensorValue);       // Print the sensor value to the Serial Monitor
intledBrightness = map(lightSensorValue, 0, 1023, 0, 255); // Map the sensor value to LED brightness
analogWrite(9, ledBrightness);          // Control LED brightness based on the sensor reading
delay(100);                             // Wait for 100 milliseconds before the next loop iteration
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/Oz0p9CI61bY

Practical 3: Program using temperature sensors

Aim:

To study the working of Temperature sensors using Arduino

Simulation Environment: TinkerCAD (Free online simulator)

Components: Arduino UNO, Temperature Sensor TMP 36

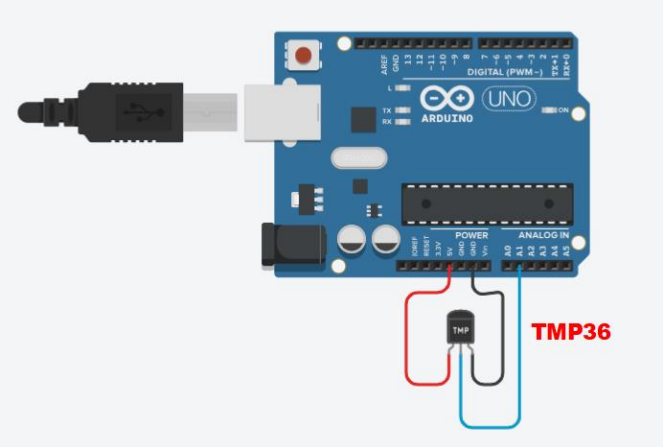
Theory:

The TMP36 is a low-cost analog temperature sensor that can be easily integrated with Arduino boards. It provides an analog voltage output that varies linearly with temperature. This practical aims to show how to measure and display real-time temperature data using a TMP36 temperature sensor and an Arduino. The temperature data will be displayed through suitable method.

The TMP36 temperature sensor is a precision analog sensor. It generates an output voltage that is linearly proportional to the Celsius temperature. It typically has three pins: VCC, GND, and OUT. The sensor's output voltage increases by 10 mV per degree Celsius. At 25°C, it outputs 750 mV.

The demonstration showcases the practical application of the TMP36 temperature sensor in conjunction with an Arduino board for real-time temperature monitoring. It highlights how to interface the sensor, read its analog output, and display the temperature information. This knowledge can be applied to various temperature-sensing applications, including weather stations, environmental monitoring, and more.

Circuit Diagram:



Pin Connections:

Arduino	TMP36 Sensor
5V	Left pin
GND	Right pin
A_1	Center pin

Code:

```
char degree = 176; // ASCII Value of Degree
const int sensor = A1;
void setup() {
  pinMode(sensor, INPUT);
  Serial.begin(9600);}

void loop() {
  int tmp = analogRead(sensor); // Read data from the sensor. This voltage is stored as a 10-bit
number.
  float voltage = (tmp * 5.0) / 1024; // Convert the 10-bit number to a voltage reading.
  float tmpCel = (voltage - 0.5) * 100.0; // Convert voltage to Celsius.
  Serial.print("Celsius: ");
  Serial.print(tmpCel);
  Serial.println(degree);
  delay(1000);
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/rB3QKd-DJNg

Practical 4: Program using Humidity sensors

Aim:

To study the working of Humidity sensors using Arduino

Simulation Environment: TinkerCAD (Free online simulator)

Components: Arduino UNO, Potentiometer (wiper)

Theory:

Potentiometer as a Sensor:
A potentiometer, often referred to as a "pot," is a variable resistor with three terminals.
It consists of a resistive track and a wiper that moves along the track. By adjusting the wiper's position, you can vary the resistance.
In this demonstration, the potentiometer is used to simulate a variable sensor input.

Arduino:
Arduino is a versatile microcontroller platform commonly used for various electronic projects.
It can read analog voltage levels from sensors, including potentiometers, and convert them into digital values for processing.

TinkerCAD:
TinkerCAD is a web-based platform for simulating and designing electronic circuits and Arduino-based projects.
It's an excellent tool for testing and prototyping virtually, even when physical components are unavailable.

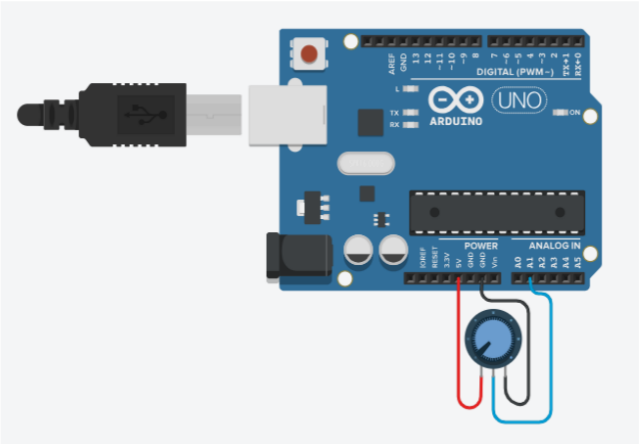
Demo Overview:
In this demo, we learn how to connect a potentiometer to an Arduino board in the TinkerCAD environment.
We understand the wiring and connections required to read variable resistance values from the potentiometer accurately.

Programming:
We see how to write the code to read and convert the analog voltage from the potentiometer into digital values and the humidity.

Practical Applications:
While the potentiometer doesn't directly measure humidity, we observe how variable sensor inputs are used in applications like volume control, dimmer switches, and other scenarios where adjustable values are required.
The demo provides hands-on experience in interfacing a potentiometer with an Arduino, which can be a valuable skill for various electronic projects.

Understanding how to read analog values and convert them into digital format is a fundamental aspect of working with sensors and input devices.
The demo serves as a practical example of using a potentiometer in an Arduino project and its potential applications in real-world scenarios.
While the potentiometer isn't a humidity sensor, this demonstration can still be educational and relevant, regarding interfacing variable sensors with Arduino.

Circuit Diagram:



Potentiometer

Pin Connections:

Arduino	Potentiometer
5V	Left pin
GND	Right pin
A_1	Center pin

Code:

```
/*
This code records the humidity using a simulated potentiometer.
The Humidity is simulated by mapping the potentiometer output into
percentages.
*/

constintanalogIn = A1; // Connect the humidity sensor to this pin
inthumiditySensorOutput = 0;

void setup() {
Serial.begin(9600);
}

void loop() {
humiditySensorOutput = analogRead(analogIn);
inthumidityPercentage = map(humiditySensorOutput, 0, 1023, 10, 70);

Serial.print("Humidity: "); // Printing out Humidity Percentage
Serial.print(humidityPercentage);
Serial.println("%");
delay(5000); // Iterate every 5 seconds
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/CFL5rtbQ95A

Practical 5: Programs using Ultrasonic Sensors

Aim:

To study the working of Ultrasonic sensors using Arduino

Simulation Environment: TinkerCAD (Free online simulator)

Components: Arduino UNO, HC-SR04 sensor

Theory:

The HC-SR04 is an inexpensive and widely used ultrasonic distance sensor module. It is often employed in various projects and applications, such as robotics, automation, and DIY electronics. The name "HC-SR04" is derived from the model or product code of this specific sensor module.

The HC-SR04 sensor utilizes ultrasonic sound waves to determine the distance between the sensor and an object. Here's how it works:

a. Ultrasonic Emission: The sensor emits a high-frequency sound wave, usually in the ultrasonic range (around 40 kHz). This sound wave is inaudible to humans.

b. Sound Wave Reflection: The emitted sound wave travels through the air until it encounters an object. When it hits the object, it bounces back towards the sensor.

c. Receiving the Echo: The sensor has a built-in receiver to detect the reflected sound wave, also known as an echo.

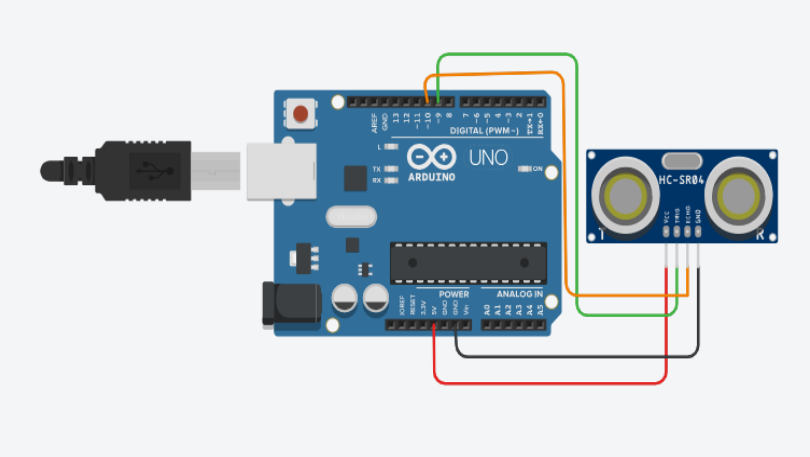
d. Calculating Distance: By measuring the time it takes for the sound wave to travel to the object and back (i.e., the time it takes for the echo to return), the HC-SR04 can calculate the distance to the object using the speed of sound in the air (approximately 343 meters per second or 1125 feet per second at room temperature).

e. Output: The sensor provides the calculated distance as an output in the form of a digital pulse or duration in microseconds that can be easily converted to distance in centimeters or inches.

This distance measuring technique is non-contact, making it suitable for a wide range of applications, including obstacle avoidance in robots, measuring liquid levels, and more. The HC-SR04 sensor is popular among hobbyists and electronics enthusiasts due to its affordability,ease of use, and compatibility with microcontrollers like Arduino and Raspberry Pi.

It typically has four pins: VCC (power supply), GND (ground), Trig (trigger), and Echo (echo signal output).

Circuit Diagram:



Pin Connections:

Arduino	HC-SR 04 Sensor
5V	V_{CC}
GND	GND
Pin 9	TRIG
Pin 10	ECHO

Code:

```
// Define the pins for the ultrasonic sensor
constinttrigPin = 9;  // Arduino digital pin for the trigger
constintechoPin = 10; // Arduino digital pin for the echo

// Variables to store the duration and distance
long duration;
int distance;

void setup() {
  // Initialize serial communication for debugging
Serial.begin(9600);
```

```
 // Define the trigger and echo pins as OUTPUT and INPUT
pinMode(trigPin, OUTPUT);
pinMode(echoPin, INPUT);
}

void loop() {
 // Trigger a pulse to the sensor
digitalWrite(trigPin, LOW);
delayMicroseconds(2);
digitalWrite(trigPin, HIGH);
delayMicroseconds(10);
digitalWrite(trigPin, LOW);

 // Measure the duration of the pulse from the echo
duration = pulseIn(echoPin, HIGH);

 // Calculate the distance based on the speed of sound
distance = duration * 0.034 / 2; // Divide by 2 because the sound travels to the object and back

 // Print the distance to the serial monitor
Serial.print("Distance: ");
Serial.print(distance);
Serial.println(" cm");

 // Add a delay between measurements
delay(1000); // 1 second
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/0VZrqKZsUQE

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.

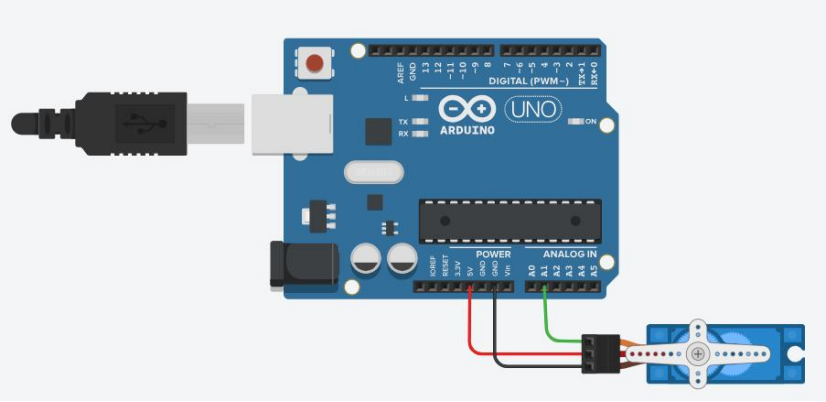
When we connect a micro servo motor to an Arduino, we typically use a library (such as the Servo library in Arduino) to control its movements.

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 the Servo library
#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)
  }
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/4cNkbFo10vI

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.

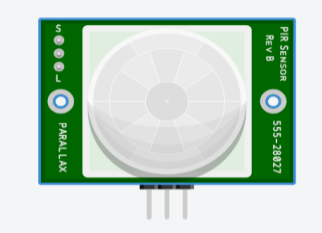
5. Output Signal: The sensor typically provides a digital output signal that can be used to trigger an alarm, turn on lights, or perform other actions based on the detected motion.

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.

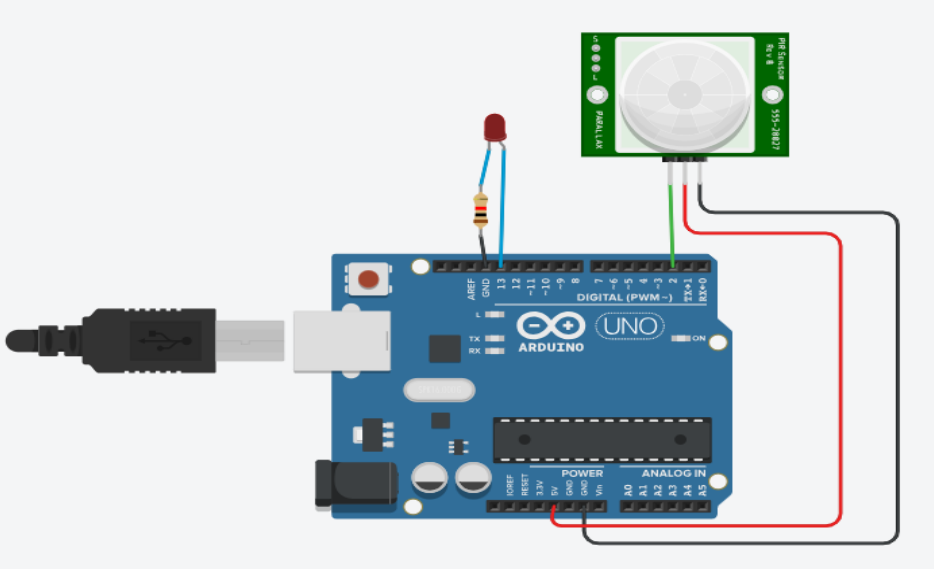
The PIR sensor has three terminals, which are listed below:



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
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/4cNkbFo10vI

Practical 8: Programs using Gas sensors

Aim:

To detect smoke/fire using Gas sensor

Simulation Environment: TinkerCAD (Free online simulator)

Components: Arduino UNO, Gas sensor, LED, resistor and Breadboard

Theory:

Gas sensors are devices designed to detect and measure the concentration of gases in the surrounding environment. They are widely used in various applications, including industrial safety, environmental monitoring, medical diagnostics, and home automation. Gas sensors play a crucial role in ensuring the safety of individuals and detecting potential hazards.

The working principle of gas sensors can vary depending on the type of sensor and the specific gas it is designed to detect.
The basic principle used in a smoke detector, whether in a real-world device or a simulated one in Tinkercad, is the change in electrical conductivity or resistance in the presence of smoke particles.
The principle can be understood through the following steps:

1. Gas Sensing Element:
   - In a real smoke detector, a specialized gas sensing element is used. This element often consists of a material that interacts with smoke particles in the air.

2. Change in Conductivity or Resistance:
   - When smoke particles are present, they interfere with the normal operation of the gas sensing element. This interference leads to a change in the electrical conductivity or resistance of the sensing element.

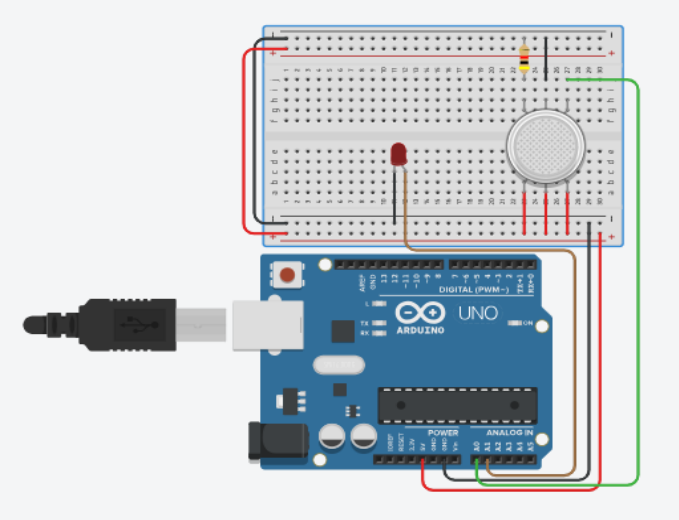
3. Voltage Divider Circuit:
   - The gas sensor is typically part of a voltage divider circuit. In the case of a simulated circuit in TinkerCAD, a variable resistor is often used to represent the gas sensor.

4. Arduino Interface:
   - The output of the voltage divider circuit is connected to an analog pin on an Arduino. The Arduino reads the analog value, which corresponds to the resistance or conductivity of the gas sensor.

5. Threshold Detection:
   - A threshold value is set in the Arduino code. If the analog value exceeds this threshold, it indicates that the resistance of the gas sensor has changed significantly, suggesting the presence of smoke.

6. Alarm Activation:
- When the threshold is surpassed, the Arduino activates an alarm signal. In the Tinkercad simulation, this is often represented by turning on an LED.

Circuit Diagram:



Arduino	Gas Sensor	LED	Resistor
5V	B1		
	B2		
	B3		
GND	H1	Cathode	Other End
	A1		One End
A0	A2		
A1		Anode	

Code:

```
int LED = A1;
constint gas = 0;
int MQ2pin = A0;

void setup() {
Serial.begin(9600);
}

void loop() {
float sensorValue,MQ2pin;
sensorValue = analogRead(MQ2pin); // read analog input pin 0

if(sensorValue>= 470){
digitalWrite(LED,LOW);
Serial.print(sensorValue);
Serial.println(" |SMOKE DETECTED");


  }
else{
        digitalWrite(LED,HIGH);
Serial.println("Sensor Value: ");
Serial.println(sensorValue);
  }
delay(1000);
}
        floatgetsensorValue(int pin){
        return (analogRead(pin));
}
```

For the Video Demonstration of the given practical click on the link below or scan the QR-Code

https://youtu.be/2G9GYcBoRQI